A STUDY OF TWO-RAIL TOTALLY SELF-CHECKING CIRCUITS

ZHI-JIAN JIANG, B.Eng.

A STUDY OF TWO-RAIL TOTALLY SELF-CHECKING CIRCUITS

BY

©ZHI-JIAN JIANG, B. ENG.

A THESIS SUBMITTED TO THE SCHOOL OF GRADUATE STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING

FACULTY OF ENGINEERING AND APPLIED SCIENCE
MEMORIAL UNIVERSITY OF NEWFOUNDLAND

FEBRUARY 1992

ST. JOHN'S NEWFOUNDLAND CANADA

ISBN 0-315-73354-3

Canada

Dedicated to my parents

Abstract

In order to overcome the limitations of conventional design techniques of totally self-checking (TSC) circuits and provide simple, convenient, and systematic design techniques, we formalize a new two-element morphic Boolean algebra — strong morphic Boolean algebra B_{SM} — and propose a new classification of checkers. Based on these, we have developed three types of universal two-rail (TR) totally self-checking (TSC) basic building blocks (BBB) — EIS BBBs, EISS BBBs, and EIIS BBBs. Besides, a group of TR-TSC multi-function BBBs has also been proposed. These BBBs, like ordinary logic gates in common digital circuits, can be easily used to implement any arbitrary combinational logic using our proposed design rules. The resulting circuit is a TR-TSC circuit.

A simple interconnection method (SIM) and an image design method (IDM) for the design of TR-TSC circuits have been proposed. The SIM is suitable for the case that the *self-testing* property can be easily achieved or verified. The IDM deals with the general case that includes complicated logic functions with a large number of inputs.

We also present a new method to design a TSC circuit with separate error-input indication (EI) and separate internal fault indication (IF). This objective has been achieved by using a new BBB — a TR-TSC decoupling BBB (DC_2).

An efficient method of diagnosing relevant error sources has been studied. With the help of decoupling circuits consisting of DC_2's, the error status of relevant inputs and outputs can be indicated. This greatly improves localizability and enhances maintainability. A totally new circuit concept named *error-confining circuit* has been introduced. TR-TSC error-confining (ECF) circuits implement given logic

functions during fault-free operation. But when any internal fault from a prescribed set of faults occurs, the circuit automatically forms several independent areas which are surrounded by isolation boundaries. Thus, the fault is confined to a special area and indicated. This property enhances localizability, maintainability and availability. A TSC double-input decoupling (DIDC) BBB, which is a key component to be used in constructing the isolation boundaries, has been developed.

The design problems of TSC sequential BBBs have also been discussed. A scheme for designing TSC D flip-flop has been proposed.

In addition, we have also developed an efficient combinational TSC checker for 1-out-of-3 code. The proposed checker uses less hardware, has fewer gate levels, and possesses a higher test capability.

Acknowledgements

I would like to express my most gratitude and appreciation to Dr. R. Venkatesan for being my supervisor and for all the valuable discussions, helpful guidance, and consistent encouragement throughout the entire period of my program.

I would also like to express my sincere thanks to the members of the School of Graduate Studies and the Faculty of Engineering and Applied Science of Memorial University of Newfoundland for admitting me into the graduate program in Electrical Engineering.

I wish to take this opportunity to acknowledge the financial support through Dr. R.Venkatesan's research grant, Graduate Fellowship and Bursary of Memorial University of Newfoundland, and Department Support of Faculty of Engineering and Applied Science, which made this work possible.

I thank the Memorial University for providing all the working facilities, particularly to the staff of computer centre of the Faculty of Engineering and Applied Science for their constant help and up-to-date equipment, as well as the staff of the Queen Elizabeth II library for their efficient interlibrary loan program.

Finally, I should always keep in mind the dedication from my wife Meng Rong and my lovely daughter Betty through their love and support.

Contents

List of Figures

List of Tables

Symbols

Φ	*Input space*
Φ_c	*Input code space*
Φ_{nc}	*Input noncode space*
Ψ	*Output space*
Ψ_c	*Output code space*
Ψ_{c1}	*Correct output code space*
Ψ_{c2}	*Incorrect output code space*
Ψ_{nc}	*Output noncode space*
AIPS	*Advanced information processing system*
B_u	*Usual Boolean algebra*
B_M	*Morphic Boolean algebra*
B_{SM}	*Strong Boolean morphic algebra*
BBB	*Basic building block*
C_n	*A two-rail code with n bits*
CDS	*Cyclic diagnostic sequences*
CEIS	*Conditional error input sensitive*
C.SEDTR	*Conditional single-error detecting operator block*
CTHS	*Complement translator with high strobe*
CTLS	*Complement translator with low strobe*
DC_2	*Relevant two-input decoupling basic building block*
DC_4	*Relevant four-input decoupling basic building block*
DEDTR	*Double-error detecting operator block*
DEIS	*Double-error input sensitive*
DIDC	*Double-input decoupling basic building block*

ECF	*Error-confining circuit*
EI	*Error-input indication*
EIS	*Error input sensitive*
EISS	*Error input semi-sensitive*
EIIS	*Error input insensitive*
FA	*Full adder*
FC	*Functional circuit*
FC-H	*Functional circuit with high-strobe valid*
FC-L	*Functional circuit with low-strobe valid*
FTMP	*Fault-tolerant multiprocessor*
IDM	*Image design method*
IF	*Internal fault indication*
MAND	*Morphic AND operator block*
MEDTR	*Two-rail multi-error detecting checker*
MF	*Multi-function*
MNOT	*Morphic complement operator block*
MUX	*Multiplexer*
MXOR	*Morphic XOR operator block*
N_2	*Two-rail two-input comparator*
N_n	*Two-rail TSC checker for a two-rail code* C_n
NMR	*N-modula. redundancy*
NSC	*Nonself-checking*
OPTR	*Odd-parity operator block*
Sb	*Strobe*
SC	*Self-checking*
SEDTR	*Single-error detecting operator block*

SIFT	*Software implemented fault tolerance*
SIM	*Simple interconnection method*
SM	*Strong morphic operation*
SMR	*Sift-out modular redundancy*
T_S	*Complete minimal test set*
TMR	*Triple modular redundancy*
TR	*Two-rail*
TSC	*Totally self-checking*
VT	*Voter*

Chapter 1

Introduction

1.1 The motivation for the research

Totally self-checking (TSC) circuits [1], [2] are highly desirable for ultrahighly reliable digital system design [3]–[6]. The superiority of TSC circuits to other digital circuits is mainly reflected on three aspects. First, temporary faults — transient faults and intermittent faults are detected; second, faults are immediately detected upon occurrence; third, software diagnostic programs are no longer necessary [3]–[6].

Considerable work has been done on the design of TSC circuits [1]–[31]. However, unlike the design of ordinary logic circuits which has not only a group of logic gates and a set of sequential logic blocks available, but also has Boolean algebra as its theoretical basis, neither general theory nor universal basic building blocks (BBBs) are available for the design of TSC circuits. Existing two-rail (TR) totally self-checking (TSC) operator blocks [9], [11]–[16], [31] are only suitable for the design of TSC checkers [9], [11]–[16], [23]. Thus, existing techniques for designing TSC circuits remain complicated, inflexible and non-systematic, and to construct a TSC circuit is still a challenging work.

1.2 Objectives and organization of the Thesis

In this thesis, developing a group of universal BBBs and formalizing a general theory for the design of TSC circuits are our two main objectives.

A brief overview on the development and major techniques of fault tolerance has been given in *Chapter 2*. In comparison to software scheme of fault tolerance, hardware scheme possesses a higher speed and reliability, especially the TSC techniques provide concurrent error detection for both temporary and permanent errors.

In *Chapter 3*, we have summed up the existing design methods for TR-TSC checkers. It shows that TR-TSC checkers not only can be independent TSC checkers to check other TSC functional circuits but also are widely used in designing other useful TSC checkers. TR-TSC checkers can be constructed by only two-input two-variable comparators (N_2), as well as by universal operator blocks.

A new Boolean algebra called strong morphic Boolean algebra B_{SM} has been formalized in *Chapter 4*. This new Boolean algebra deals with two-element morphic variables and possesses both the properties of usual Boolean algebra B_u and conventional morphic Boolean algebra B_M. We also have proposed a new classification of checkers. According to the new classification, five types of checkers have been defined.

Based on B_{SM} and new definitions of checkers, four groups of universal TR-TSC BBBs have been developed in *Chapter 5*. These four groups of BBBs are error-input sensitive (EIS) BBBs, error-input semi-sensitive (EISS) BBBs, error-input insensitive (EIIS) BBBs, and a group of TR-TSC multi-function (MF) BBBs. These universal BBBs can be easily applied to the design of TR-TSC circuits, and are particularly efficient for TR-TSC functional circuits.

In *Chapter 6*, we have described two design techniques to construct TR-TSC circuits using the proposed BBBs and previous operator blocks. A simple interconnection method (SIM) has been proposed and it is suitable for the case where the *self-testing* property can be easily achieved or verified. For the general case, we have proposed a design technique called image design method (IDM). This method can be used to design TSC circuits which implement any given combinational logic functions. It enhances the testability, and the self-testing property of resulting circuits is easy to be verified. A new block called TR-TSC decoupling BBB (DC_2) has been developed. DC_2 is very useful and efficient for designing a new class of TSC checkers which have a separate error-input indication (EI) and a separate internal fault indication (IF). Decoupling techniques for relevant error indication variables has been studied. The decoupling technique provides a new way to locate faulty units and greatly improves maintainability. The concepts of isolation boundary and error-confining (ECF) circuit have been introduced. Properly using the proposed BBBs, a new generation of TSC circuits which have the capability of confining errors can be achieved. An auxiliary block called TR-TSC double-input decoupling (DIDC) BBB has been developed. This block is particularly useful for constructing the isolation boundaries.

In *Chapter 7*, we have proposed an efficient TSC combinational checker for 1-out-of-3 code. The checker itself is not a TR-TSC checker. However, the achievement of the checker is based on the principles of TR-TSC checker and morphic space theory.

In *Chapter 8*, this thesis is concluded and recommendations for further research are discussed.

Chapter 2

An overview of fault tolerance

2.1 The importance of fault tolerance

As the theoretical basis of fault tolerance, the history of fault-tolerant computing dates back to the early 1940's when the evolution of computing systems reached the level of complexity of the relay computers developed by Harvard University and Bell Telephone Laboratories [32].

Designers of the first electronic computer, the ENIAC, were likewise aware of the extreme reliability requirements associated with the operation of a complex computing system. In the words of H. H. Goldstine, one of the principal developers of the ENIAC [33]:

> *"To gain some rough measure of the magnitude of the risks (in undertaking the development of the ENIAC) we should realize that the proposed machine turned out to contain over 17000 tubes of 16 different types operating at a fundamental clock which issued a signal every 10 μs. Thus, once every 10 μs an error would occur if a single one of the 17000 tubes operated incorrectly; this means that in a single second*

there were 1.7 billion (=1.7 × 10^9) chances of a failure occurring and in a day (=100000 s) about 1.7 × 10^{14} chances. Put in other words, the contemplated machine had to operate with a probability of malfunction of about a part in 10^{14} in order for it to run for 12 h without error. Man had never made an instrument capable of operating with this degree of fidelity or reliability, and this is why the undertaking was so risky a one and the accomplishment so great."

The reliability requirements of the ENIAC were met, for the most part, through careful selection, pretesting, and use of each component. However, as early as 1946 it became clear that, as the physical and logical complexity of electronic computers increased, other precautions would have to be taken to enhance the reliability and availability of computing systems.

In 1948, J. von Neumann proposed that the components of a computer should be viewed as having a nonzero probability of failure and suggested triple modular redundancy (TMR) as a means of improving system reliability [34]. In von Neumann's belief, the correct way to design fault-tolerant computing systems was to utilize redundant components in some systematic manner, the TMR scheme being but one way of doing this.

It was the establishment of the IEEE Computer Society Technical Committee on Fault-Tolerant Computing in May 1970 that both theoretical and practical aspects of fault-tolerant computing began to grow steadily [32], [35].

Fault-tolerant computing has been defined as "the ability to execute specified algorithms correctly regardless of hardware failures, total system flaws, or program fallacies" [35]. Basically the technology of fault-tolerant computing encompasses

theory and techniques of fault and error detection and correction, modeling, analysis, synthesis, and architecture of fault-tolerant systems and their evaluation [35]. It can be classified into three areas:

- The design and analysis of fault-tolerant circuits and systems;
- The diagnosis and testing of digital circuits and systems;
- The validation of programs or "software reliability".

With the recent revolutionary changes in circuit technology, more and more VLSI and ULSI chips are used as the core and auxiliary components and aimed at achieving high speed, more functions, and broader fields of applications. These require that modern computers have ultrahigh reliability, ultrahigh availability, reduced life-cycle costs, and long-life applications [5]. As a prerequisite, high quality chips must be used in building a reliable computer; therefore, we still can not build a commercially feasible computer depending only on these chips. The highly-integrated chip doubles the complexity of its internal circuits. Also, the chip itself becomes susceptible to a more diverse variety of failures. Besides internal opens and shorts and bonding failures, there are other fault modes, such as bridging faults, stuck-at faults, and crosspoint faults. Some defective cells exist when a chip has just been made, and some sprout up with the wear-out course. The uncontrollability and unobservability of the highly-integrated chip make it impossible to diagnose these faulty units promptly. In order to solve these problems, various fault-tolerance measures are employed in highly reliable computers [3]–[6].

Fault tolerance in a highly reliable digital system is achieved through redundancy in hardware, software, information, and/or computation. Fault-tolerance strategies consist of the following elements [36].

- Masking: Dynamic correction of generated errors.
- Detection: Detection of an error – a symptom of a fault.
- Containment: Prevention of error propagation across defined boundaries.
- Diagnosis: Identification of the faulty module responsible for a detected error.
- Repair/reconfiguration: Elimination or replacement of a faulty component, or a mechanism for by-passing it.
- Recovery: Correction of the system to a state acceptable for continued operation.

For short-term ultrareliable operation, where no time is available for off-line fault diagnosis and repair, a static or passive configuration of elements is designed to mask a given maximum number of faults [36].

Dynamic redundancy, on the other hand, involves the switching of modules or rerouting of communications as faults occur. The faulty components are detected, diagnosed, and repaired or replaced [36].

In a hybrid approach, a static base configuration masks a given number of faults, while faulty modules are detected and replaced within the configuration. Hybrid redundancy is desirable for long-term ultrareliable applications in which the probability of multiple faults is high [36].

High availability applications do not necessarily require continuous error-free operation, although database and other critical resources must be protected and contained within replaceable modules, rather than masked. System operation is then degraded or halted to perform diagnosis, reconfiguration or repair, and recovery [36].

2.2 Common schemes of fault tolerance

As we know, the reliability enhancement of computing systems is achieved through two fundamentally different approaches. The first approach is called fault prevention (also known as fault intolerance) and the second is fault tolerance. In the traditional fault prevention approach, the objective is to increase the reliability by a priori elimination of all faults. Since this is almost impossible to achieve in practice, the goal of fault prevention is to reduce the probability of system failure to an acceptably low value. In the fault tolerance approach, faults are expected to occur during computation but their effects are automatically counteracted by incorporating redundancy into a system so that valid computation can continue even in the presence of faults. These facilities consist of more hardware, more software or more time, or a combination of all these; they are redundant in the sense that they could be omitted from a fault-free system without affecting its operation.

Fault tolerance is not a replacement but rather a supplement to the most important principles of reliable system design: (a) use the most reliable components (however, cost constraints often preclude their use); and (b) keep the system as simple as possible, consistent with achieving the design objectives.

Redundancy can be implemented in static, dynamic, or hybrid configurations [3]-[6].

2.2.1 Static redundancy

Static redundancy, also known as "masking redundancy", uses extra components, and the effect of a faulty component is masked instantaneously. Two major

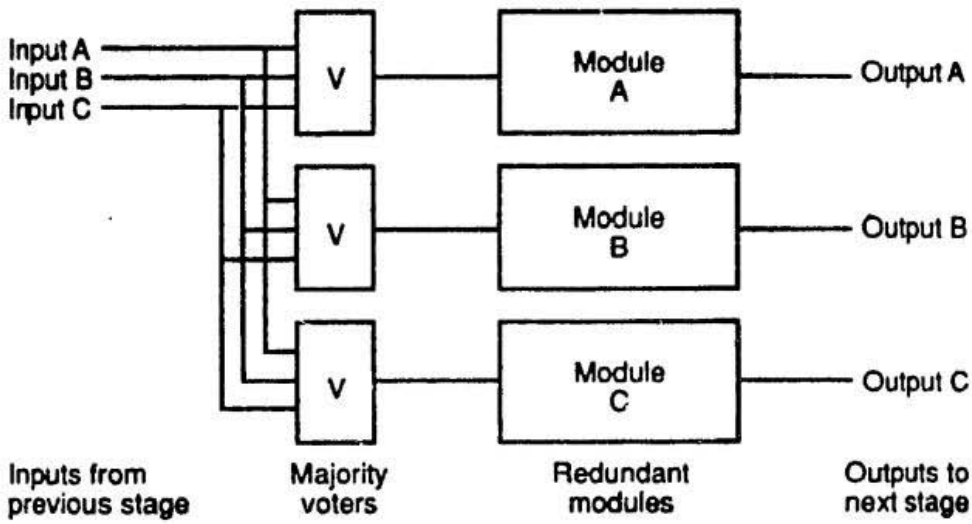

Figure 2.1: Triplicated voters and modules forming one triple modular-redundant stage of system, with voting at module inputs copied from Ref [36].

techniques employed to obtain fault masking are the triple modular redundancy (TMR) and the use of error correcting codes [4], [36]. For example, a TMR system [36] is shown in Figure 2.1.

Continuous operation is often provided by using the majority vote of the outputs of three or more identical modules, masking failure of the minority. Triple modular redundancy (TMR) has been used extensively in ultra-reliable systems for aerospace and industrial applications, with two out of three votes masking single-module failures.

Coding is the most widely developed mechanism for error detection, correction

and masking in digital system, typically requiring less redundancy than other error detection, correction and masking schemes [37]. A code's error detection and correction properties are based on its ability to partition a set of 2^n n-bit words into a code space of 2^m words and a noncode space of $2^n - 2^m$ words. For most codes, each word comprises m-bits of information and $k = n - m$ check bits. Each code is designed so that a given number of errors transforms a code-space word into a word in the noncode-space. Errors are detected by decoding circuits that identify any word outside the code-space. Error correction is performed by more extensive decoding that uniquely associates a noncode-space word with the original code word transformed by the errors.

Hamming distance between the words of the code space determines the capability of error detection or correction of a given code-space. The most common words include simple parity checks to detect errors in buses, memory, and registers. Parity-based Hamming codes detect and correct errors in memory; cyclic redundancy checks and other cyclic codes detect and correct errors in communication channels and disk storage; m-out-of-n codes detect errors in microprogram control stores and other ROMs; and arithmetic codes detect errors originating within arithmetic logic units (ALU). Unidirectional error control codes have found a real-time application in VLSI microprocessors, where the bus line area increases as the processor word length increases. Since these lines connect circuit elements, line faults or defects seriously affect LSI chip yield and reliability [3]–[6], [36].

Figure 2.2 shows a bus line circuit that can mask single "0" errors. The decoder G consists of AND gates g_0 to g_3, corresponding to code words V_0 to V_3 in code C. Because each gate has transistors at the bus line where the element in the code word is "1", a gate is only activated by receipt of the corresponding code word.

In general, when "0" errors change the code word Y to Y', any other code word, say X, in C has to satisfy the condition $X\overline{Y'} \neq 0$. This causes the bus line circuit to work correctly. This condition shows that there are one or more cases where X has 1 at the position where Y' has 0. From this, it can be easily proved that a code C with $\Delta = t + 1$ can mask t asymmetric errors [37].

As implementation costs for error-control coding continue to decrease, the development of cost-effective, low-level techniques may offset the need for massive high-level redundancy. Therefore, the major challenge of the future is developing an integrated design framework where we can study the various trade-offs between low-level and high-level redundancy techniques.

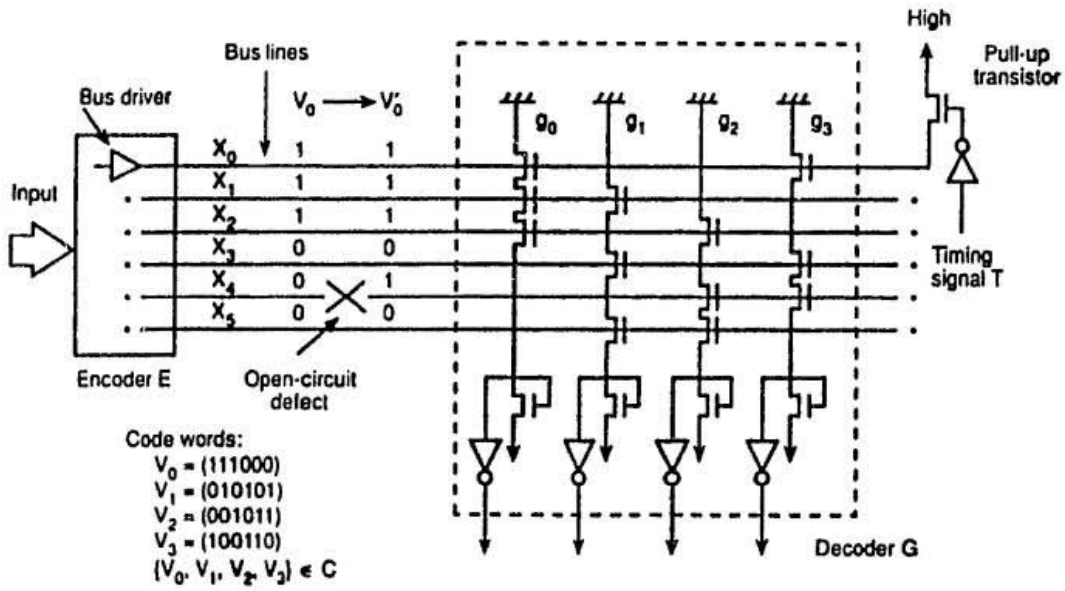

Figure 2.2: Circuit for masking single, unidirectional "0" errors on bus line copied from Ref [37].

2.2.2 Dynamic redundancy

Dynamic redundancy is also known as "Standby Redundancy". In the dynamic redundancy, spare modules are switched into the system when working modules break down [3]–[6].

Figure 2.3 illustrates the concept of dynamic redundancy [4]. The system consists of S + 1 modules but only one operates at a time. If a fault is detected in the working modules, it is switched out and replaced by a spare. Thus, dynamic redundancy requires consecutive actions of fault detection and fault recovery.

Recovery is defined as the continuation of system function after the incidence of an error in data integrity.

The detection of fault in the individual modules of a dynamic system can be achieved by using one of the following techniques [4]:

1. Periodic tests;
2. Self-checking circuits;
3. Watchdog timers.

In periodic tests, the normal operation of the function module is temporarily suspended and a test routine is run to determine if faults are present in the module. A disadvantage of this technique is that it cannot detect temporary faults unless they occur while the module is tested.

Self-checking circuits are designed so that they either provide correct output or indicate the presence of a fault in a module during normal operation.

Watchdog timers are set to certain values at pre-established points, called check-

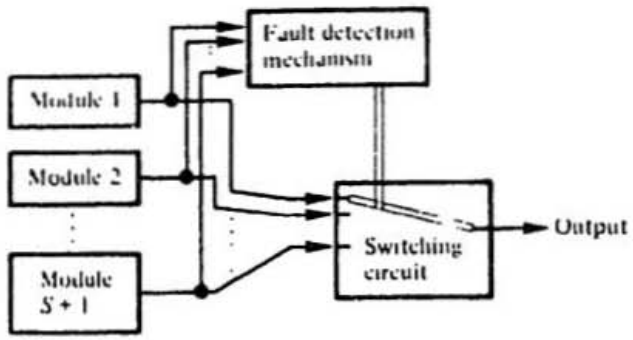

Figure 2.3: Dynamic redundancy scheme with S spares copied from Ref [4].

points, in the program executed by a module. A timer at a particular checkpoint counts down while the module performs its function, and is normally reset before the next checkpoint is reached. However, a software bug or a hardware fault will prevent the program from resetting the timer. The timer then issues an interrupt command which causes automatic switch over to a spare module.

Dynamic redundant systems can also be classified to cold-standby system and hot-standby system [4], [36].

In a cold-standby system, one module is powered up and operational, the rest are not powered. Replacement of a fault module by a spare is effected by turning off its power and powering a spare. In a hot-standby system, all the modules are powered up and operating simultaneously. If the output of all modules are the same, the output of any arbitrarily selected module can be taken as the system output. When a fault is detected in a module, the system is reconfigured so that the system output comes from one of the remaining modules.

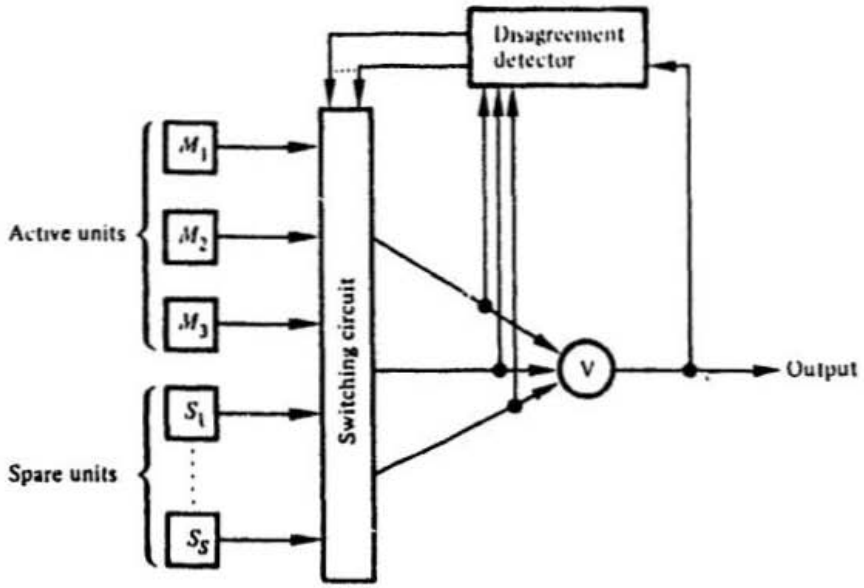

Figure 2.4: Hybrid redundancy system with TMR scheme and S spare modules copied from Ref [4].

2.2.3 Hybrid redundancy

Hybrid redundancy combines the static and the dynamic redundancy approaches. It consists of an NMR system (in general) with a set of spare modules. When one or less than $n = (N-1)/2$ of the NMR modules fails, it is replaced by a spare and the basic NMR operation can continue [4], [36]. A hybrid scheme is shown in Figure 2.4 [4].

Redundancy can also be implemented by other hardware redundant techniques such as self-purging redundancy, and sift-out modular redundancy (SMR) [4].

Replacement units can be either *hot* or *cold* [36]. A hot spare concurrently performs the same operations as the module it is to replace, needing no initialization where it is switched into the system. A cold spare is either not powered or used for other tasks, requiring initialization when switched into the system. System designer must weigh the cost of unused spare against that of initialization time

when deciding between hot and cold spares.

2.2.4 Time redundancy

Time redundancy is commonly used in the detection and correction of errors caused by temporary faults [4]. It involves the repetition or rollback of instructions, segments of programs or entire programs immediately after a fault is detected. The rollback operation requires that a program restarts processing from the last checkpoint, where all the information relevant to the successful execution of the program beyond the checkpoint is stored. If a fault is temporary, rollback the program to a checkpoint should allow successful recovery. However, if the fault is permanent, the fault detection mechanism will be activated again and an alternative recovery method should be attempted.

2.2.5 Software redundancy

Redundancy, which is used to achieve fault tolerance in hardware, has not found wide application in software. The main problem is that it is not possible to quantify the expected improvement in reliability that can be achieved by using additional software. Chen and Avizienis have suggested the idea of *N-version programming* for providing fault tolerance in software [4].

In the N-version programming approach, a number of independently written programs for a given function are run simultaneously, results are obtained by voting upon the outputs from the individual programs. In general, the requirement that the individual programs should provide identical outputs is extremely stringent. Therefore, in practice *sufficiently similar* output from each program is regarded as equivalent. However, this increases the complexity of the voter. In addition to its

ability to tolerate design faults, N-version programming is also capable of masking certain categories of temporary hardware faults.

2.2.6 Fail-soft operation

If a faulty system has ability to continue to operate at an acceptable but reduced level of performance, when the faulty modules are disconnected from the system and the rest of the system are reconfigured, this ability is known as fail-soft operation [4].

In order to achieve the capability of fail-soft operation, a system must have a distributed architecture, a comprehensive fault detection capability, the ability to achieve both logic and power isolation between functional modules, and the ability to reconfigure itself to operate as efficiently as possible without a fault module.

The space shuttle computer complex is an example of this strategy. It uses four processors with majority voting for critical operations. Voting continues after one failure, but a second failure ends voting and a single processor performs all remaining operations.

2.2.7 Practical fault-tolerant systems

By the end of the 1960's nearly all of the basic forms of fault-tolerant architecture to be found in later designs had been built and experimented with (e.g., triplication with voting, duplication and comparison, self-checking units, and backup sparing). These concepts were refined and adapted to more modern hardware and software technology in subsequent computers [4], [38], [39].

Two very advanced research machines were developed to the same specifica-

tions by the same NASA sponsor, and were built and tested as prototypes: the fault-tolerant multiprocessor (**FTMP**) and software implemented fault tolerance (**SIFT**). Simplified block diagrams of the two architectures are shown in Figure 2.5.

Both systems execute three copies of a program in different hardware and vote the results to mask faults but they do it in quite different ways. All processors in the **FTMP** are clock synchronized and voting is done by hardware. Processors in **SIFT** use independent clocks, and voting and synchronization are carried out by software.

In the **FTMP** structure, a set of processors and memories are connected to five redundant buses through special redundant bus guardian circuits. Processors and memory modules can be dynamically assigned to be a member of a group of three processors and three memories which will run the same computation (designated a triad). This is done by commanding their associated bus guardians to communicate over specially assigned buses. The guardian circuits in the processors vote on the three copies of data arriving from their assigned memories, and conversely the memories guardian vote on information from their assigned processors. If a bus, processor, or memory fails, there will still be two valid copies of information at each voter, and the fault will be masked, allowing the triad to continue. When such a failure occurs, a different triad can sense the condition and reconfigure the affected triad by sending commands to bus guardians to assign a new processor, memory, or bus to the affected triad.

The **SIFT** computers are totally connected. Each computer can broadcast a message over a serial line to dedicated buffers in all the other computers. The computers operate with unsynchronized hardware clocks, and synchronization oc-

curs by a software, voting process. Each computer contains a synchronous software executive, and software voting procedures. Periodically, the computers exchange messages containing their views of the time, and develop a consensus as its value. As user processes are scheduled in a time-synchronous fashion, they are executed at approximately (but not exactly) the same time and send their results to the other processors where a software voting procedure is invoked to mask faults. If a computer fails and generates disagreeing outputs, the other two ignore it.

In practice the **FTMP** architecture has two major advantages over **SIFT** in dedicated real-time control applications. It runs faster than **SIFT** because its voting is done by hardware. The **SIFT** computers use a significant percentage of their processing time running the software voting and synchronization programs. More importantly, the fault-tolerant features of **FTMP** are nearly software transparent. Nearly any software executive can be run on **FTMP** with fault recovery procedures written to run under it. (Remember, the triads will continue to operate under fault conditions until a reconfiguration procedure is invoked). **SIFT**, on the other hand, is constrained to using its custom synchronous executive which implements the fault-tolerance features.

Although the relative hardware cost of a highly fault-tolerant computer may be several times that of a non-fault-tolerant machine, hardware prices have dropped an even greater relative amount making fault tolerance cost-effective for a large number of applications.

Advanced information processing system (AIPS)

Figure 2.6 is a structure of the advanced information processing system (AIPS) [38]. A group of processing sites is connected through switching nodes to a redun-

dant inter-communication structure which behaves like a triply redundant bus, but which can be circuit-switched over different paths to provide physical damage tolerance. Each processing site may be a fault-tolerant multiprocessor (**FTMP**), a triplicated (TMR) fault-tolerant processor (FTP), a duplicated pair of processors, or a single non-redundant machine. Each site has a local clock which synchronizes computers at that site, but clocks are not synchronized between sites. Hardware voting is done throughout the system. With a site containing triplicated (TMR) processors, voting is straightforward because the processors are clock-synchronized and are executing identical programs. Voting of triplicated data sent between processing sites with different local clocks requires a hardware synchronization operation, but the data skew can probably be kept small and hardware voting is still feasible. This design recognizes the needs for selective redundancy. In a complex system, not all tasks are sufficiently critical to justify triplicating their processors. Thus, duplex and single processors can be included.

In general, the hardware implementation of a fault-tolerant systems is naturally achieved at several levels based on functions provided by specific subsystems. It contains redundant components and recovery mechanisms which may be employed in different ways at different levels. For example, at the highest level, a distributed system may recover from a failed computer by shifting its computations to other machines. At the next lower level, a single computer may be capable of replacing a faulty memory module with a spare and switching to alternative communication channels to circumvent a failed part, but may not be able to recover when a short occurs on the local memory-processor bus or when its power supply fails. At a lower level, the memory modules may be capable of replacing defective RAM chips with spares, or the chips may contain redundancy and be capable of tolerating certain

failures but not others.

2.3 Self-checking (SC)

For fault detection, modules at all levels (computers, logic modules, or on-chip redundancy), fall between two basic types. At one extreme are circuits which can detect internal faults concurrently with normal operation — we will call them *self-checking* (SC) — and at the other extreme are modules which have no internal fault detection capability which will be designated *nonself-checking* (NSC) [38]. When used in a redundant partition, SC modules can be operated singly, since faults will be detected, if an external recovery mechanism can substitute a spare module for the one which has failed. NSC modules must be duplicated and operated two-at-a-time with outputs compared for fault detection and, three-at-a-time and voted if a faulty module is to be identified quickly (or if transient faults are to be located).

Self-checking circuits offer a number of advantages, the most obvious of which is the immediate detection of errors. Another is the capability of detecting errors caused by transient failure. Further, self-checking circuits are becoming more attractive with the advancement in VLSI technology [3]–[6].

A self-checking computer can be developed at an approximately 10 percent increase in hardware complexity [40]. The reason for this relatively low cost is that the majority of a computer's logic is memory which due to its regular structure can be designed to detect faults with a few extra bits per word. Irregular logic must often be duplicated, but this makes up a small percentage of many modern machines.

An important characteristic of this methodology is the fact that self-checking

checkers have been developed which signal faults in the checking circuitry as well as in the operational circuits. This largely solves the problem of "who checks the checker?" Checking signals are implemented as complementary *morphic* pairs which alternate between values (10) and (01) when no error exists. Upon detecting an error in the circuits being checked or in the checking circuits, these signals take on values (11) or (00) indicating a fault has occurred. A reduction circuit has been developed so that a number of these complementary pairs from individual checkers can be reduced to a single self-checked pair which serves as a master fault indicator [38]. A general structure of TSC circuit is shown in Figure 2.7.

A TSC circuit is composed of two parts – a TSC functional circuit and a TSC checker [6]. The TSC functional circuit implements given functions during fault-free operation. The TSC checker monitors the outputs of functional circuit, internal error propagation paths of the TSC functional circuit, as well as the inputs of the TSC functional circuit.

If any error input is applied to its inputs or any internal fault from a prescribed set of fault occurs in the functional circuit or in the checker itself, the output of the checker will indicate this fault during normal operation.

There are many works on the design of self-checking circuits, especially on the design of TSC checkers such as parity code checkers, duplication checkers, parity prediction checkers, m-out-of-n codes checkers, Berger codes checkers, and residue code checkers [3]–[6]. These checkers have already been applied in an adaptive manner to some functional circuits, for example, adders, multipliers, decoding circuits, data path circuits, and the like.

Unlike the design of non-redundant logic circuits, there is no simple, conve-

nient, systematic method available for designing TSC circuits, particularly for TSC functional circuits.

Moreover, the structure of conventional TSC circuits is very complicated. This leads to a low localizability. We need a more complicated decoding circuit if the correction scheme is employed.

Consequently, it is desirable to provide a general method to design TSC circuits which makes the design of TSC functional and TSC checkers consistent and efficient.

2.4 Concluding remarks

The development of the technology of fault tolerance has been driven by high requirements of modern military system, astronauts, aerospace missions, commercial activities, communications, etc. Fault tolerance can be achieved at different levels. Although NMR scheme is widely applied to construct highly reliable systems, it is desirable that modular units have the capability of concurrent error detection so that they can signal any fault occurrence. Since TSC circuits possess this desirable property, they have been intriguing many researchers. However, the design of TSC circuits is still a pending problem. We hope that the design of TSC circuits could be done as easily as that of common digital circuits. Thus, it is highly desirable to develop a group of universal basic building blocks and formalize a set of design methods for achieving TSC circuits.

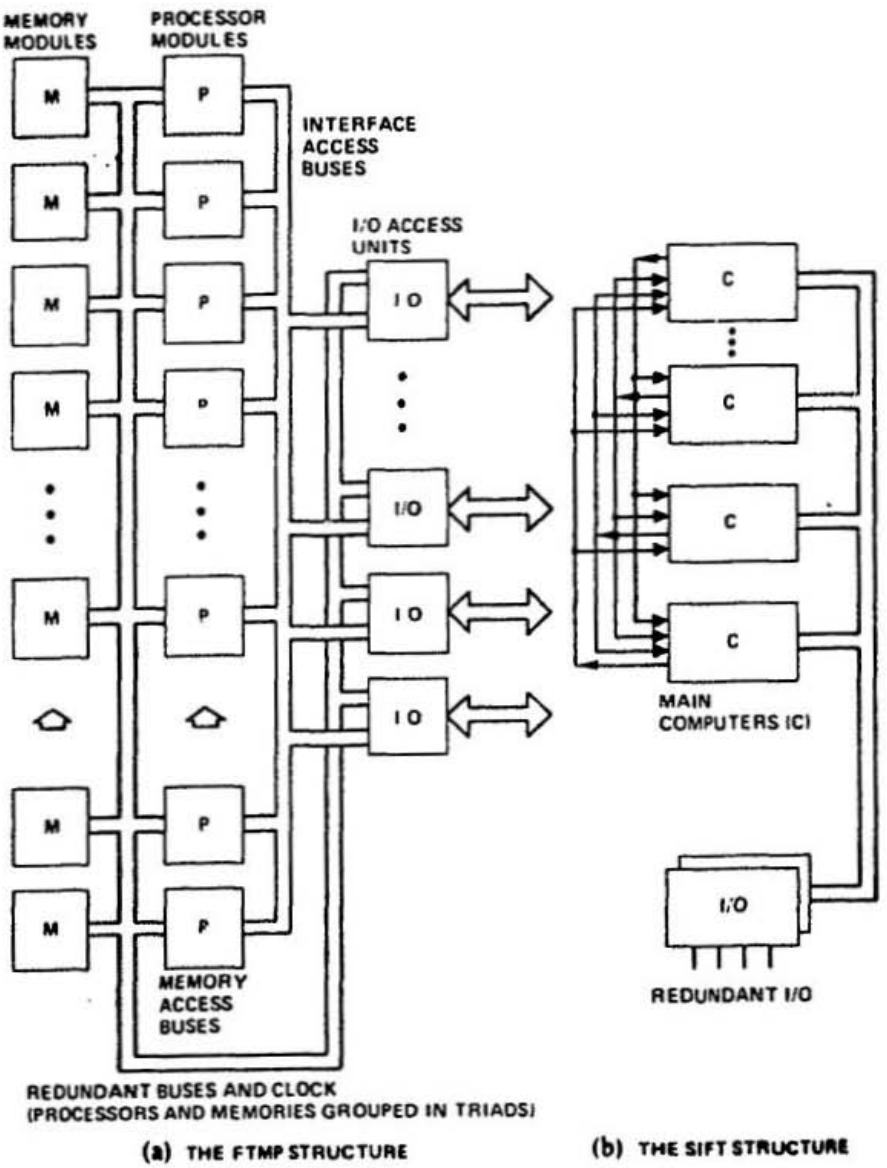

Figure 2.5: Simplified block diagrams of the **FTMP** and **SIFT** flight control computers copied from Ref [38].

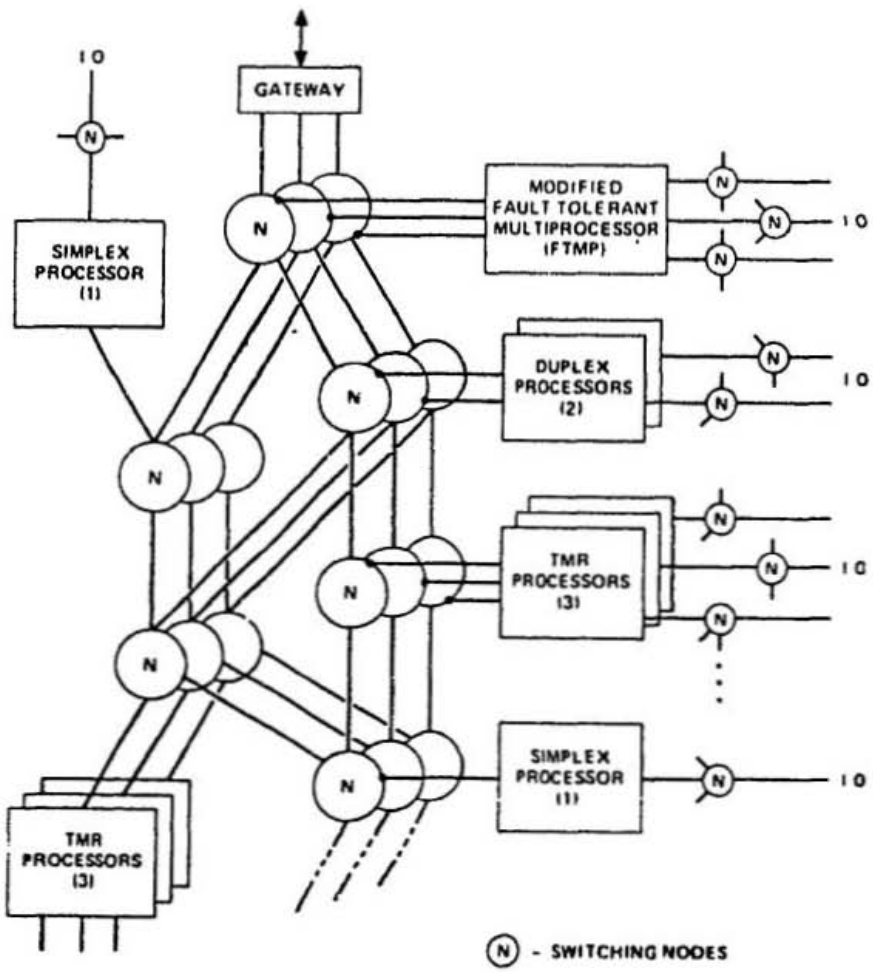

Figure 2.6: Advanced information processing system (AIPS) copied from Ref [38].

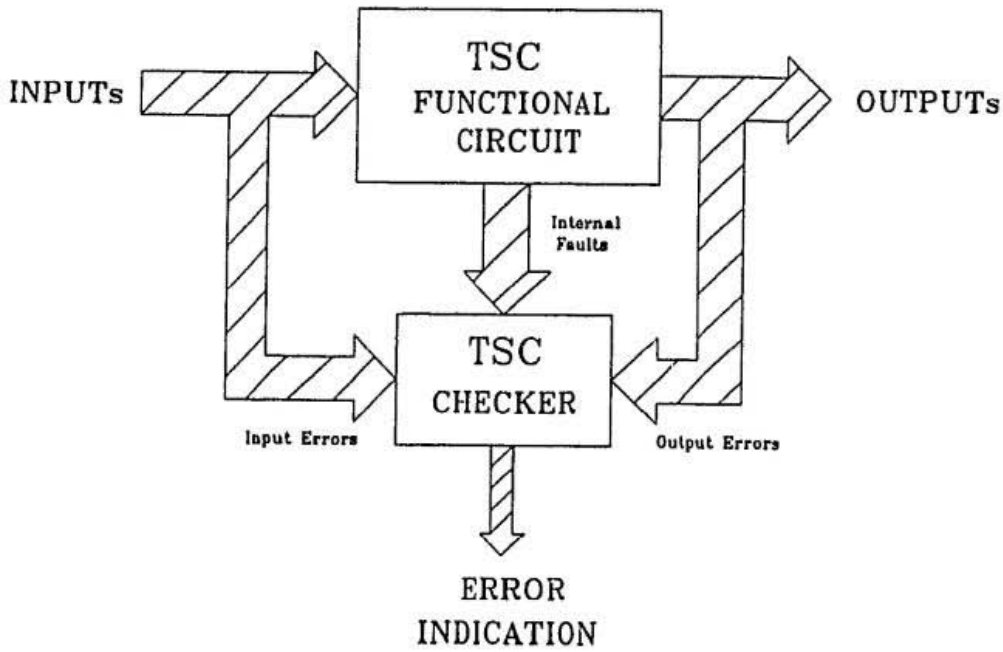

Figure 2.7: A general structure of TSC circuit.

Chapter 3

Principles of two-rail totally self-checking checkers

3.1 Preliminaries

3.1.1 Failure, fault, and error

When applied to digital systems, the terms of failure, fault, and error have different meanings [3]–[6], [36].

Failure: Failure denotes inability of an element to perform its designed function because of errors in the element or its environment, errors are in turn caused by various faults [36].

Fault: A fault is an anomalous physical condition which may or may not cause a failure. Causes include design errors, manufacturing problems, damage fatigue, or other deterioration and external disturbances.

Error: An error is a manifestation of a fault in a system, in which the logical state of an element differs from its intended value [36].

A fault in a system does not necessarily result in an error. An error occurs

only when a fault is *sensitive*. A fault is referred to as latent if it has not yet been sensitized in the system. The term soft is often applied to errors that persist after the originating fault disappears. Once corrected, soft errors usually leave no damage in the system.

3.1.2 Modeling of faults

In general, the effect of a fault is represented by means of a model, which represents the change that the fault produces in circuit signals [3]–[6], [36].

Stuck-at-faults: It assumes that a fault in a logic gate results in one of its inputs or the output being fixed to either a logic 0 (stuck-at-0) or a logic 1 (stuck-at-1). The stuck-at-fault model offers good representation for the most common types of failures, e.g. short-circuits (shorts) and open-circuits (opens) in many technologies.

Bridging fault: A bridging fault occurs when two leads in a logic network are connected accidentally and *wired logic* is performed at the connections. Two types of common bridging faults are input bridging fault and feedback bridging fault.

Stuck-open faults: Stuck-open faults are a peculiarity of CMOS digital integrated circuits; they are not equivalent to classical stuck-at faults. The major difference between the stuck-at faults and the stuck-open faults is that the former leaves the faulty gate as a combinational circuit, but the latter turns it into a sequential circuit.

The error modes which are possible to turn up can also be categorized into symmetric errors and asymmetric errors; independent bit errors and physically clustered bit errors; transient faults or intermittently occurring bit-errors and permanent faults. Transient faults are non-recurring temporary faults. Intermittent

faults are recurring faults that reappear on a regular basis.

3.1.3 Totally self-checking (TSC) concept

The concept of totally self-checking was first proposed by Carter and Schneider [1], and was formally defined by Anderson and Metze [2], [7] as follows:

Definition 1 : *A circuit is code-disjoint if it maps code inputs into code outputs and noncode inputs into noncode outputs during fault-free operations.*

Definition 2 : *A circuit is self-testing if for every fault from a prescribed set, the circuit produces a noncode output for at least one code input.*

Definition 3 : *A circuit is fault secure if for every fault from a prescribed set, the circuit never produces an incorrect code output for code inputs.*

Definition 4 : *A circuit is totally self-checking if it is both self-testing and fault secure.*

Definition 5 : *A circuit is a TSC checker if it is both code-disjoint and totally self-checking.*

TSC checkers are mainly used to monitor the outputs of TSC functional circuits and produce an error indication when one or more noncode inputs are received or any internal fault from a prescribed set occurs. They can be achieved at different levels, such as transistor level [30], logic gate level [3]-[6], functional unit level [11]-[14] and system level [16]. They also can be implemented with different technologies, such as NMOS/CMOS implementation [30], PLA implementation

[4], combinational logic implementation [3]-[6], and sequential logic implementation [10].

There have been many proposed TSC checkers which employ simple, useful, and efficient codes such as Parity codes, Berger codes, m-out-of-n codes, Hsiao code, Hamming codes, low-cost arithmetic codes, two-rail codes, etc. [3]-[6].

3.2 Two-rail (TR) TSC checkers

Two-rail (TR) means that each TR variable is represented by a pair of ordinary variables. A TR TSC checker is a TSC circuit which its input variables and output variables are two-rail variables [5], [6]. Each pair of TR variables has complementary values during normal operation. A TR-TSC checker itself can be an independent checker to monitor a TSC functional circuit. It also can be a part of a TSC checker [5], [6], [19]. One of its useful functions is of converting N error indication variables into one error indication variable for an observable error indication output. With this important property, TR-TSC checkers, especially two-input two-rail comparator N_2 [41], are often used in designing other useful TSC checkers. For example, A TSC Berger code checker which uses TR-TSC checkers as its reduction circuits [20] is shown in Figure 3.1.

3.3 The design of TR-TSC checkers using two-input two-rail comparator N_2

3.3.1 Tree structure with maximum depth

Let C_n denote a two-rail code with n bits and N_n denote a checker for C_n. In general, C_n does not always contain 2^n codewords. Checker N_n has n pairs of

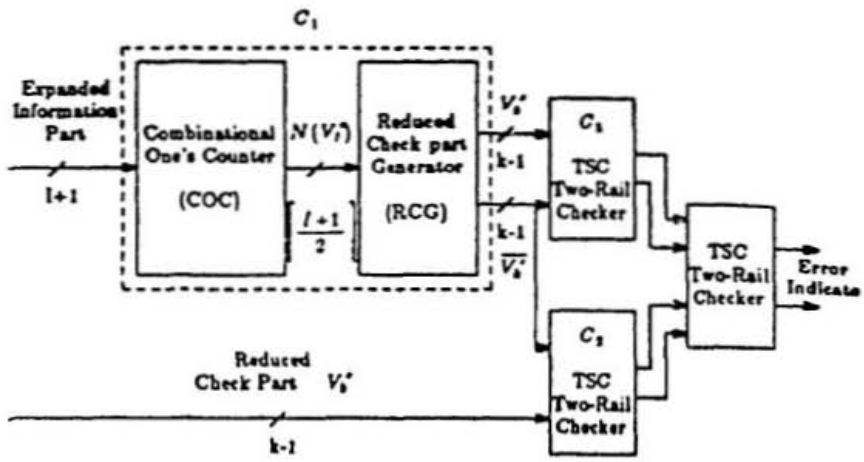

Figure 3.1: A TSC Berger code checker which uses TR-TSC checkers as its reduction circuits copied from Ref [20].

inputs (X_{i1}, X_{i2}) $(1 \leq i \leq n)$ and a pair of outputs (z1, z2). Input (X_{i1}, X_{i2}) and output (z1, z2) are represented by X_i and Z, respectively. The prescribed set of faults are single stuck-at faults, which are on concern here. Any fault in N_n can be transient, intermittent, or permanent [19].

A two-input two-rail TSC comparator [41] is shown in Figure 3.2. It is a basic TR-TSC block which is widely used to build other multi-input TR-TSC checkers. Let N_2 denote this two-input TR-TSC comparator. If (01) represents logic 0 and (10) for logic 1 in normal condition, the N_2 is equal to modulo-2 adder. Multi-input TR-TSC checkers can be implemented by interconnecting N_2 to form multilevel trees of arbitrary size [3]-[6], [19]. For example, a tree with eight input variables formed by interconnecting seven N_2 checkers is shown in Figure 3.3.

Since each block is fault secure for all unidirectional multiple faults, and self-

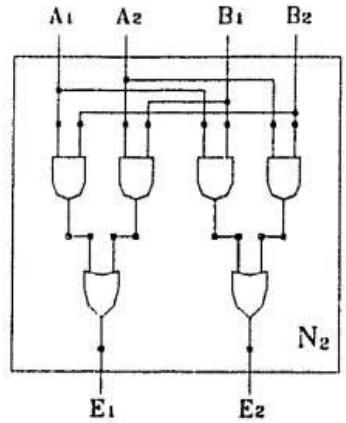

Figure 3.2: A two-input two-rail comparator N_2 [41].

testing provided that it receives the complete minimal test set T_S[1]. Thus, the resulting circuit is a TSC checker. Proofs for the following theorems can be found in [19].

Theorem 1 *[19]: N_n is a TR-TSC checker for C_n if and only if: i) every fault in N_n is detected by some word in C_n; ii) it maps code inputs into code outputs and noncode inputs into noncode outputs.*

In general, 2^n test patterns are sufficient to diagnose such multiple-input trees if each TR block has no more than n input pairs [19]. However, all these patterns may not be applied to the multiple-input tree circuit during normal operation. This is because two-rail checkers are usually placed at the output of the circuit under check; that is, they are embedded, and hence a restricted number of patterns may be given to the checkers. Even for this situation, some techniques have been proposed that

[1]A complete minimal test set (T_S) consists of T_1, T_2, T_3, and T_4, where $T_1 = \{(01),(01)\}$, $T_2 = \{(01),(10)\}$, $T_3 = \{(10),(01)\}$, and $T_4 = \{(10),(10)\}$.

satisfy the self-testing conditions [19].

Theorem 2 *[19]: Let all of the 2^n two-rail codewords be applied to N_n which is a tree circuit consisting of only N_2's. The number of codewords which give one pattern from T_S on the two pairs of input line of every N_2 in N_n is equal to 2^{n-2} $(n \geq 2)$.*

Theorem 3 *[19]: Any n-input tree circuit consisting of only N_2's is an TR-TSC checker for C_n if $k \geq (3 \times 2^{n-2} + 1)$.*

In case $k < (3 \times 2^{n-2} + 1)$, a circuit consisting of only N_2's cannot always become a TR-TSC checker. In this case, however, the probability that the circuit is a TR-TSC checker is expected to be high [19].

There are many other design methods for constructing TR-TSC checkers, such as *Tree Structure With Minimal Depth* [19], *Decision Graph Design* [19], and *Auxiliary Input Technique* [42]. These design methods have their particular advantages [19], [42].

3.4 The design of TR-TSC checkers with universal operator blocks

TR-TSC checkers can be achieved not only using N_2's but also using universal operator blocks [9], [11]–[16].

3.4.1 Two-element morphic Boolean algebra

Two-element morphic Boolean algebra was formalized by Carter *et al* [9] and it is the theoretical basis of the design of TR-TSC checkers using universal operator

blocks.

We define:

(A). The usual Boolean algebra as:

$$B_u = \{0, 1, \{*\}\} \tag{3.1}$$

where $\{*\}$ is the set of usual logic operators.

(B). The two-element Boolean algebra as:

$$\begin{aligned} B_M &= \{\{(e_1, \bar{e}_2), (\bar{e}_1, e_2)\}, \{(e_1, e_2), \\ &\quad (\bar{e}_1, \bar{e}_2)\}, \{*_M\}\} \end{aligned} \tag{3.2}$$

where $\{*_M\}$ is the set of morphic logic operators.

We have the morphism:

$$M(e_1, e_2) = \begin{cases} 1, & e_1 \neq e_2 \\ 0, & e_1 = e_2 \end{cases} \tag{3.3}$$

where $e_1, e_2 = 0$ or 1.

This morphism relates the two-element Boolean algebra B_M to the usual Boolean algebra B_u and the correspondence between them is:

$$M(A_i *_M A_j) = M(A_i) * M(A_j) \tag{3.4}$$

where $A_i, A_j \in \{(0,0), (0,1), (1,0), (1,1)\}$.

3.4.2 Factorization technique

Carter *et al* [9] proposed a set of universal operator blocks consisting of three basic blocks — MNOT, MAND, and MXOR blocks. Their internal constructions

are shown in Figure 3.4, Figure 3.5, and Figure 3.6, respectively. They formalized a factorization technique consisting of three steps to achieve a TR-TSC checker which implements any given morphic Boolean function using their proposed operator blocks. These three steps are [9]:

(a) Factorization step

Take the subpolynomial S consisting of the largest number of terms containing a common variable, say A_i, and replace it by $A_{i1}(P_1)$, where P_1 is a Boolean polynomial such that $A_{i1}(P_1) \equiv S$.

Apply the above step recursively to the remaining terms of the polynomial G until the remaining polynomial has no terms with variables in common.

(b) Parenthesis-removal step

If any Boolean polynomial within the innermost parenthesis has an odd number of terms and is immediately followed by two right parentheses, then delete the outer parenthesis.

Apply (a) and (b) alternatively until no more factorization or parenthesis removal is possible.

(c) Complementation step

In the form which cannot be subjected to further factorization or parenthesis removal, replace the substructure $1 + P$ by P'.

A self-testing structure implementing the morphic Boolean function is obtained as follows from the final parenthesized form resulting from the above algorithm. The complementation is implemented by MNOT block. The + is implemented by MXOR block. The AND structure is implemented by MAND block.

Theoretically, any arbitrary morphic logic expression can be implemented by TR-TSC checkers using these blocks and the factorization technique; therefore, complex morphic logic expressions with multiple inputs and outputs are difficult to implement since it is hard to transform given complex morphic logic expressions into testable structures of specified XOR-parenthesized forms under normal conditions. The verification of the self-testing property for a developed morphic circuit is difficult [9].

A design example using Factorization Technique to achieve a testable structure [9] is described below. Let

$$\begin{aligned} G &= A_7 + A_1A_2 + A_7A_8A_9 + A_7A_8A_9A_{10} + \\ &\quad A_7A_8A_9A_{11} + A_1A_2A_3A_4A_5 + A_1A_2A_4A_5A_6 \end{aligned}$$

Recursively applying *Factorization Step* and *Parenthesis-Removal Step*, we have

$$G = A_7(1 + A_8A_9(1 + A_{10} + A_{11})) + A_1A_2(1 + A_4(A_5(A_3 + A_6)))$$

Finally applying *Complementation Step*, we have the expression as

$$G = A_7(A_8A_9(A_{10}' + A_{11}))' + A_1A_2(A_4(A_5(A_3 + A_6)))'.$$

Figure 3.7 is an implementation of the given expression.

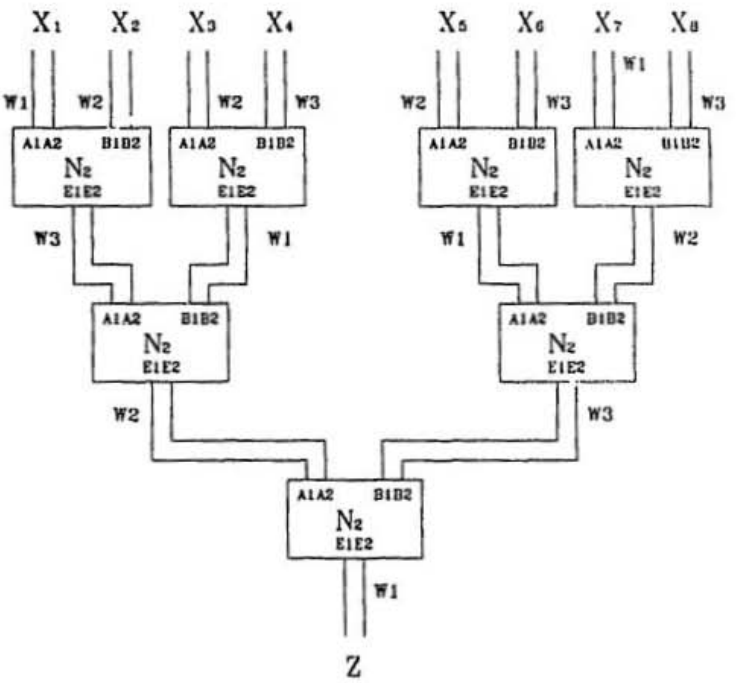

Figure 3.3: A eight-input two-rail code checker.

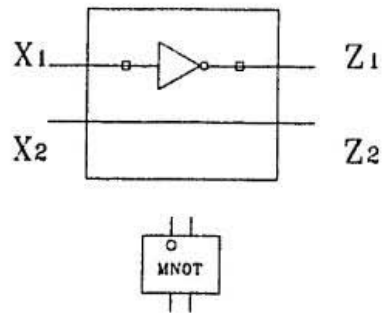

Figure 3.4: A MNOT operator block.

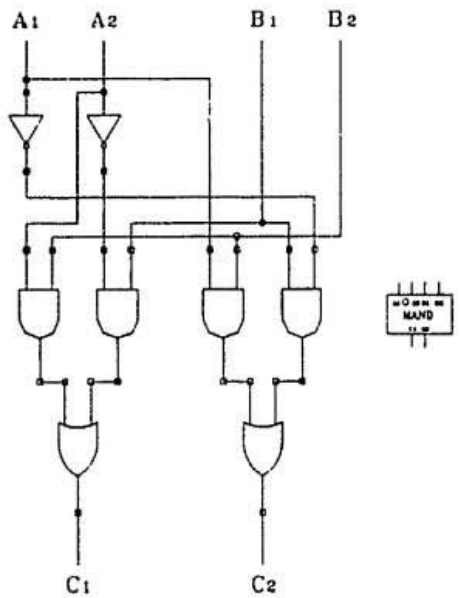

Figure 3.5: A MAND operator block.

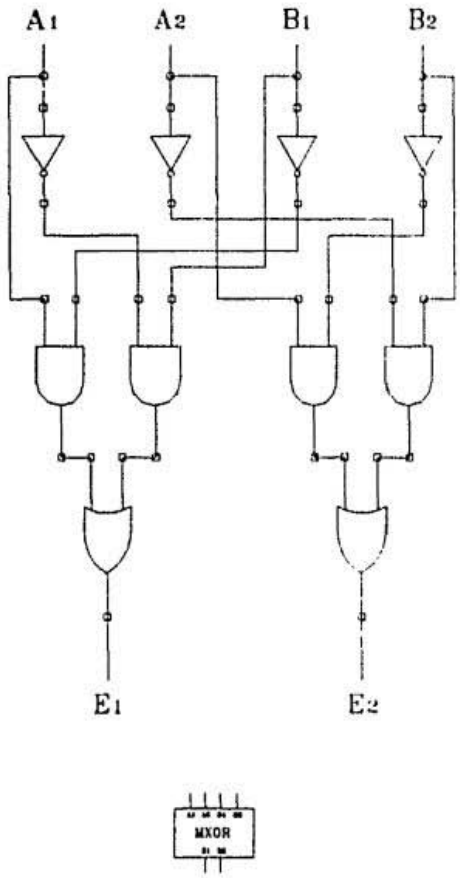

Figure 3.6: A MXOR operator block.

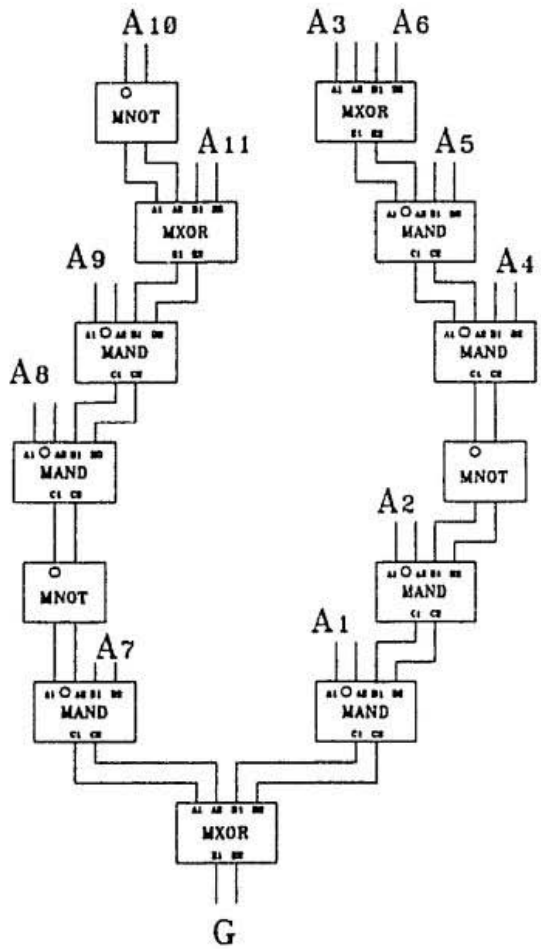

Figure 3.7: A design example with Factorization Technique.

3.4.3 Multiple error detection TR-TSC checkers

Until now, we have discussed various methods for the design of TR-TSC checkers. Although they can deal with different cases which have multiple inputs, and large amount, medium amount, as well as small amount of codewords for given two-rail code C_n, they are incapable of detecting multiple errors. As independent TSC checkers, these TR-TSC checkers have no ability to locate fault sources but indicate fault occurrence.

Based on the two-element Boolean algebra B_M, Gaitanis proposed a universal set of TR-TSC operator blocks and formalized a general technique to design multiple error detecting TR-TSC checkers [11]–[16].

The new universal set of operator blocks is composed of a single-error detecting two-rail operator (SEDTR), an odd-parity two-rail operator(OPTR), a conditional single-error detecting two-rail operator (C· SEDTR), and a double-error detecting two-rail operator (DEDTR). The combinational circuits which are designed using these blocks are TSC circuits if every internal fault propagates to a set of observable outputs. In comparison with the previous set of operator blocks, this set can be easily used in implementing complex morphic logic expressions. The self-testing property of the resulting circuits can be easily proved with cyclic diagnostic sequences CDS. The internal constructions of SEDTR, OPTR, C·SEDTR, and DEDTR are shown in Figure 3.8[2], Figure 3.9, Figure 3.10, and Figure 3.11, respectively. [b]

[2]SEDTR has the same internal construction as N_2

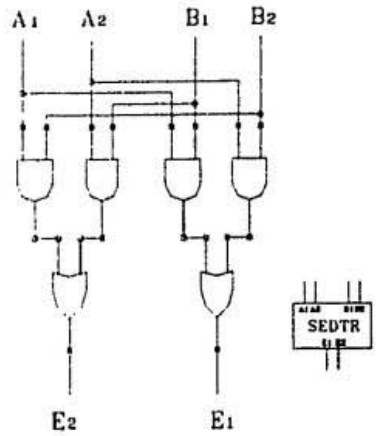

Figure 3.8: The structure of SEDTR operator block.

Design of multi-input TR-TSC double-error checkers

Suppose that we have an n_1-input DEDTR checker with single and double error indication outputs S_1^{n1} and D_1^{n1}, and an n_2-input DEDTR checker with single and double error indication outputs S_2^{n2} and D_2^{n2}. Then, multi-input DEDTR checkers can be designed according to a recursive formula as follows [14]:

$$D_3^n = D_1^{n1} D_2^{n2} \cdot (S_1^{n1} + S_2^{n2}) \tag{3.5}$$

$$S_3^n = S_1^{n1} S_2^{n2} \tag{3.6}$$

The modular structure of a multi-input TR-TSC double-error checker [14] is shown in Figure 3.12.

Design of multi-input TR-TSC multi-error checkers

Let n two-rail inputs be denoted by A_i, $i = 1, 2, \cdots, n$. Let n separate error indication outputs be denoted by T_n^e, $e = 1, 2, \cdots, n$ which indicate up to e input errors.

The multi-input TR-TSC multi-error detecting checkers called MEDTR checker can be constructed according to the following recursive functions [15]:

$$T_{r+1}^e = T_r^e(T_r^{e-1} + A_{r+1}) \tag{3.7}$$

$$T_r^e + A_{r+1} = T_r^e + T_r^{e-1} + A_{r+1} \tag{3.8}$$

where $T_r^e > T_r^{e-1}$.

Since the TR-TSC multi-error detection checkers can provide multiple error information, they are particularly useful and efficient in designing TSC SEC/DED circuits [11]-[16].

3.5 Concluding remarks

A TR-TSC checker can be an independent checker to monitor a TSC functional circuit or a key part of another TSC checker. Various design techniques are available for the design for TR-TSC checkers. According to the structure of TR-TSC checkers, there are two main methods to design TR-TSC checkers. One is to use only N_2's to construct TR-TSC checkers. The other is to use universal operator blocks to build TR-TSC checkers. Based on the functions of TR-TSC checkers, TR-TSC checkers can also be classified to error-indication-only checkers and multiple error detection checkers. Since TR-TSC multi-error detection checkers provide multiple error information, it brings new prospective applications for TR-TSC checkers.

The techniques discussed in this chapter are not suitable for the design of TR-TSC functional circuits. In the following chapters, we will introduce new theories and techniques to design TR-TSC functional circuits using universal set of basic building blocks (BBB).

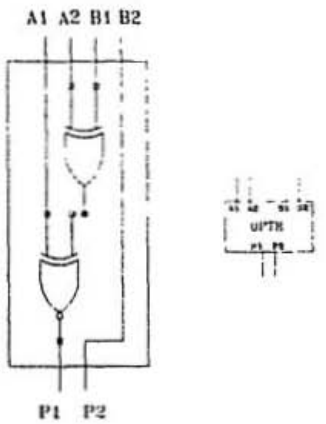

Figure 3.9: The structure of OPTR operator block.

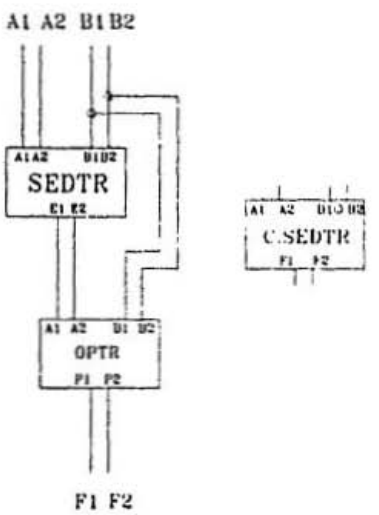

Figure 3.10: The structure of C.SEDTR operator block.

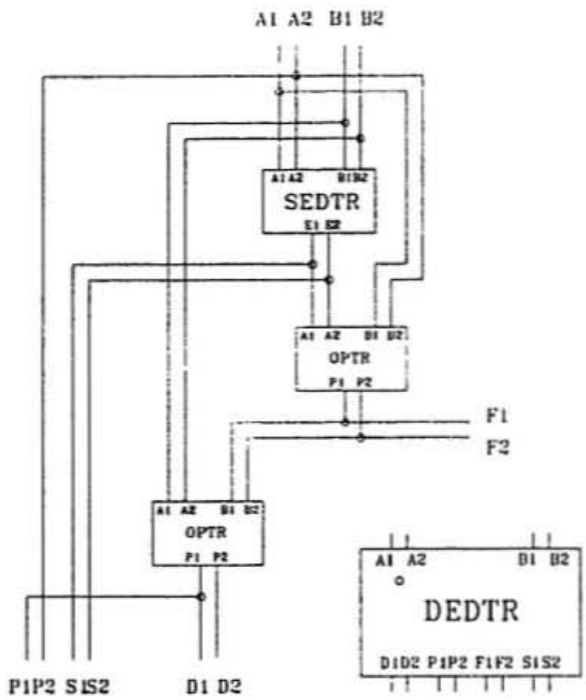

Figure 3.11: The structure of DEDTR operator block.

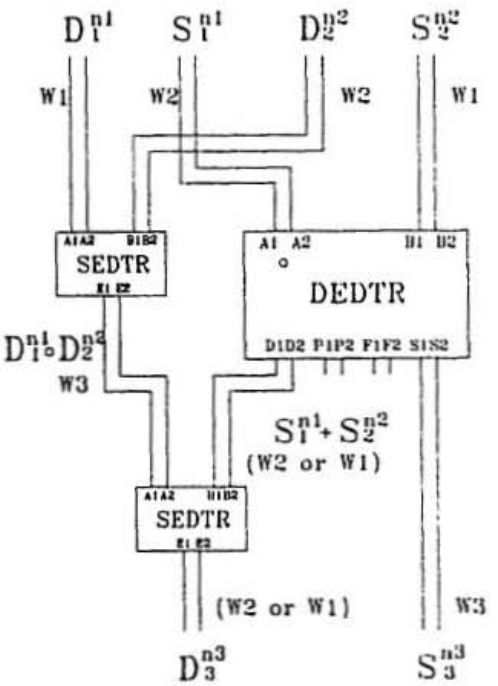

Figure 3.12: The modular structure of a multi-input TR-TSC double-error checker [14].

Chapter 4

Strong morphic Boolean algebra and a new classification of checkers

Existing universal sets of operator blocks are only suitable for the design of TR-TSC checkers [9], [11]-[16], [19] which implement conventional morphic functions with error indication variables. Conventional morphic Boolean algebra B_M formalizes two-element morphic logic [9] for error indication variables; therefore, it does not describe ordinary logic operations of usual Boolean algebra B_u under normal condition. Thus, the theories and methods of the design of TR-TSC checkers using existing universal sets of operator blocks cannot be directly applied to TR-TSC functional circuits. The design of TR-TSC functional circuits is still based on trial and error. For example, a TR-TSC full adder (FA) [31] is shown in Figure 4.1. This TR-TSC adder has been successfully achieved; now, how about the next design task for another TR-TSC functional circuit? Consequently, it is necessary to establish new theory and provide simple, convenient and systematic design techniques for the design of TSC circuits, especially for TSC functional circuits.

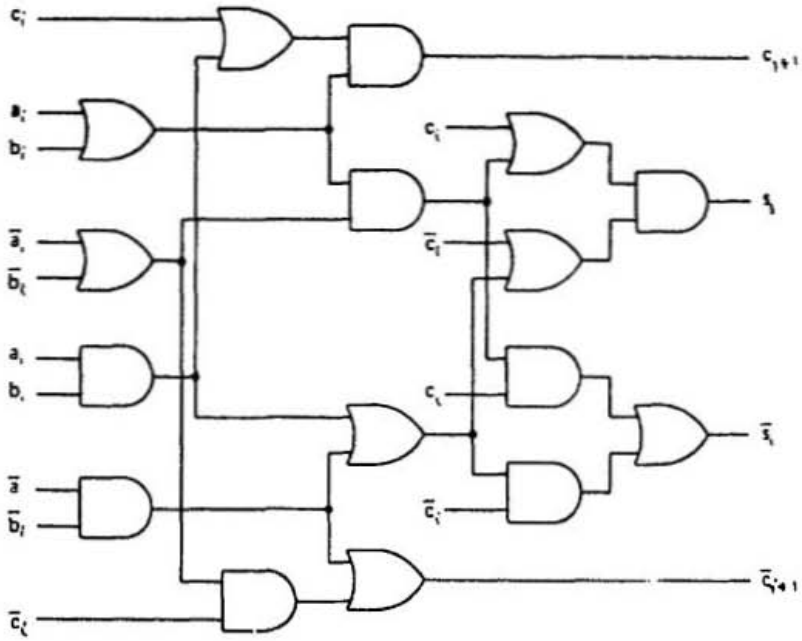

Figure 4.1: A two-rail TSC full adder copied from Ref [31].

4.1 Strong morphic Boolean algebra (B_{SM})

Two-element morphic Boolean algebra B_M discussed in *Section 3.4.1* formalizes two-element morphic logic for error indication. In B_M, (01) and (10) are equivalent for normal state; (00) and (11) are equivalent for fault state. There is no difference between (01) and (10) under normal condition. This leads that B_M does not conduct ordinary logic operations of usual Boolean algebra B_u under normal condition. In order to overcome the limitations of B_M, we introduce a new Boolean algebra called two-element strong morphic Boolean algebra B_{SM}.

We define *two-element strong morphic Boolean algebra* as:

$$B_{SM} = \{\{(e_1, \bar{e}_2), (\bar{e}_1, e_2)\}, \{(e_1, e_2), (\bar{e}_1, \bar{e}_2)\}, \{*_{SM}\}\} \tag{4.1}$$

where e_1, $e_2 = 0$ or 1, $\{*_{SM}\}$ is a set of strong morphic logic operators.

The strong morphism is formalized as:

$$M_S(e_1, e_2) = \begin{cases} 1_M, & e_1 \neq e_2 \\ 0_M, & e_1 = e_2 \end{cases} \tag{4.2}$$

and

$$1_M = \begin{cases} 1, & e_1 = 1,\ e_2 = 0 \\ 0, & e_1 = 0,\ e_2 = 1 \end{cases} \tag{4.3}$$

1_M — morphic logic 1 for normal state; 0_M — morphic logic 0 for fault state; 1 — usual logic 1, 0 — usual logic 0.

This new morphism relates the strong morphic Boolean algebra B_{SM} to usual Boolean algebra B_u. The correspondence between them is:

$$M_S(A_i *_M A_j) = M(A_i *_{SM} A_j) = M_S(A_i) * M_S(A_j) \tag{4.4}$$

where A_i and A_j are two-element error indication variables, $\{*_M\}$ is a set of operators in B_M, and $\{*\}$ is a set of operators in B_u.

Definition 6 *: The state space and operators $*_{SM}$,*

$$B_{SM} = \{\{(e_1, \bar{e}_2), (\bar{e}_1, e_2)\}, \{(e_1, e_2), (\bar{e}_1, \bar{e}_2)\}, \{*_{SM}\}\} \tag{4.5}$$

form a Boolean algebra termed strong morphic Boolean algebra.

Theorem 4 *: There is a natural correspondence between functions $g(a_1, a_2, \cdots, a_n)$ in $B_u = \{0, 1, \{*\}\}$, and functions $G(A_1, A_2, \cdots, A_n)$ in B_{SM}.*

Proof: For any logic operation from $\{*\}$ of B_u, there exists one and only one operation from $\{*_{SM}\}$ in B_{SM}. This correspondence is given in (4.4). Consequently, the natural correspondence between B_u and B_{SM} exists. □

The most prominent advantage of B_{SM} is that it relates the state (01) of two-element variable to usual logic 0 and (10) to usual logic 1 while it still possesses all the properties of B_M.

The natural correspondence between B_u and B_{SM} only formalizes the behaviour of B_{SM} in its intrinsic state space – code space. The extrinsic state space – noncode space also needs investigation.

4.2 A new classification of checkers

According to the conventional definition, a circuit is called a checker if it maps code inputs into code outputs and noncode inputs into noncode outputs during fault-free operation. However, this definition is not enough to describe the entire properties of TSC checkers in a TSC space. We should study the TSC space and give a new classification of checkers.

4.2.1 A TSC state space

A TSC space is associated with two spaces – input space Φ and output space Ψ. The input space Φ consists of code space Φ_c and noncode space Φ_{nc}. The output space Ψ consists of code space Ψ_c and noncode space Ψ_{nc}. That is

$$\Phi = \Phi_c \cup \Phi_{nc} \tag{4.6}$$

$$\Psi = \Psi_c \cup \Psi_{nc} \tag{4.7}$$

Ψ_c can be further distinguished by Ψ_{c1} and Ψ_{c2}. That is

$$\Psi_c = \Psi_{c1} \cup \Psi_{c2}$$

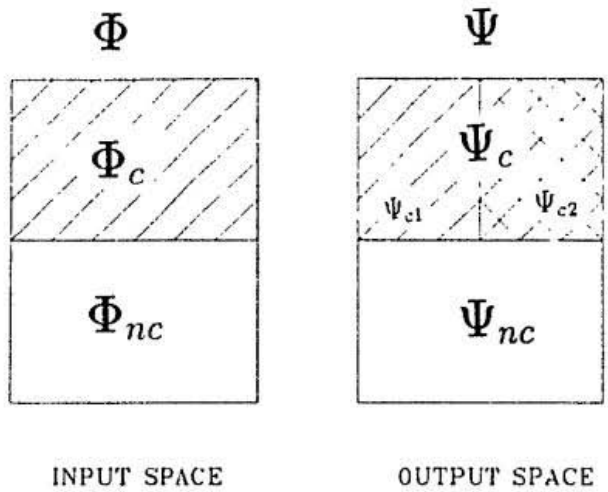

Figure 4.2: A TSC state space.

where Ψ_{c1} stands for correct code output space during normal operation and Ψ_{c2}, which is generated by noncode inputs, for incorrect code output space (disregarded code space). Thus, $\Psi_c = \Psi_{c1}$ (Ψ_{c2} is null) when inputs are code words; Ψ_c is Ψ_{nc}, or equals to Ψ_{c2} (Ψ_{c1} is null) when inputs are noncode words. Ψ_{c1} and Ψ_{c2} cannot co-exist as non-null sets.

The TSC state space is illustrated in Figure 4.2.

4.2.2 A new classification of checkers

Let X, Y, Z represent two-element error indication variables, and X, Y for input variables, Z for output variable. Based on the partition of TSC space and the behaviours of a checker in intrinsic and extrinsic state spaces, we define the following five types of checkers.

Type I checker

Type I checker is the conventional checker. The formal definition is given as:

Definition 7 : *A checker is a Type I checker if and only if:*

i) When $X,\ Y \in \Phi_c$, *then*

$$X, Y \xmapsto{G_I} Z = G_I(X, Y) \in \Psi_{c1} \tag{4.8}$$

ii) When either $X \in \Phi_c$ *and* $Y \in \Phi_{nc}$ *or* $X \in \Phi_{nc}$ *and* $Y \in \Phi_c$ *or* $X, Y \in \Phi_{nc}$, *then*

$$X, Y \xmapsto{G_I} Z = G_I(X, Y) \in \Psi_{nc} \tag{4.9}$$

G_I - *Type I mapping.*

Type II checker

Type II checker is defined as:

Definition 8 : *A checker is a Type II checker if and only if:*

i) When $X,\ Y \in \Phi_c$, *then*

$$X, Y \xmapsto{G_{II}} Z = G_{II}(X, Y) \in \Psi_{c1} \tag{4.10}$$

ii) When either $X \in \Phi_c$ *and* $Y \in \Phi_{nc}$ *or* $X \in \Phi_{nc}$ *and* $Y \in \Phi_c$, *then*

$$X, Y \xmapsto{G_{II}} Z = G_{II}(X, Y) \in \Psi_{nc} \tag{4.11}$$

iii) When both $X, Y \in \Phi_{nc}$, *then*

$$X, Y \xmapsto{G_{II}} Z = G_{II}(X, Y) \in \Psi_{c2} \tag{4.12}$$

G_{II} - *Type II mapping.*

Type III checker

Type III checker is defined as:

Definition 9 : *A checker is a Type III checker if and only if it meets:*

i) When $X,\ Y \in \Phi_c$, *then*

$$X, Y \overset{G_{III}}{\longmapsto} Z = G_{III}(X,Y) \in \Psi_{c1} \tag{4.13}$$

ii) When either $X \in \Phi_c$ *and* $Y \in \Phi_{nc}$ *or* $X \in \Phi_{nc}$ *and* $Y \in \Phi_c$, *or both* $X, Y \in \Phi_{nc}$, *then*

$$X, Y \overset{G_{III}}{\longmapsto} Z = G_{III}(X,Y) \in \Psi_{c2} \tag{4.14}$$

G_{III} - *Type III mapping.*

Type IV checker

Type IV checker is defined as:

Definition 10 : *A checker is a Type IV checker if and only if:*

i) When $X,\ Y \in \Phi_c$, *then*

$$X, Y \overset{G_{IV}}{\longmapsto} Z = G_{IV}(X,Y) \in \Psi_{c1} \tag{4.15}$$

ii) When either $X \in \Phi_c$ *and* $Y \in \Phi_{nc}$ *or* $X \in \Phi_{nc}$ *and* $Y \in \Phi_c$, *then*

$$X, Y \overset{G_{IV}}{\longmapsto} Z = G_{IV}(X,Y) \in \Psi_{c2} \tag{4.16}$$

iii) When both $X, Y \in \Phi_{nc}$, *then*

$$X, Y \overset{G_{IV}}{\longmapsto} Z = G_{IV}(X,Y) \in \Psi_{nc} \tag{4.17}$$

G_{IV} - *Type IV mapping.*

Type V checker

Type V checkers are conditional checkers which have conditional inputs and non-conditional inputs. We still consider two inputs case. Type V checker is defined as:

Definition 11 : *A checker is a Type V checker if and only if:*

i) When $X,\ Y \in \Phi_c$, *then*

$$X, Y \xmapsto{G_V} Z = G_V(X, Y) \in \Psi_{c1} \tag{4.18}$$

ii) Let X *be a conditional input,*

(a) When $X \in \Phi_c$, $Y \in \Phi_{nc}$, *then*

$$X, Y \xmapsto{G_V} Z = G_V(X, Y) \in \Psi_{nc} \tag{4.19}$$

(b) When $X \in \Phi_{nc}$, $Y \in \Phi_c$, *then*

$$X, Y \xmapsto{G_V} Z = G_V(X, Y) \in \Psi_{c2} \tag{4.20}$$

iii) Similarly, let Y *be a conditional input,*

(a) When $X \in \Phi_{nc}$, $Y \in \Phi_c$, *then*

$$X, Y \xmapsto{G_V} Z = G_V(X, Y) \in \Psi_{nc} \tag{4.21}$$

(b) When $X \in \Phi_c$, $Y \in \Phi_{nc}$, *then*

$$X, Y \xmapsto{G_V} Z = G_V(X, Y) \in \Psi_{c2} \tag{4.22}$$

iv) When both $X, Y \in \Phi_{nc}$, *then*

$$X, Y \xmapsto{G_V} Z = G_V(X, Y) \in \Psi_{c2} \tag{4.23}$$

G_{IV} – *Type IV mapping.*

According to the new definitions above, for two-input case, we know that Type I checkers are error-input sensitive (EIS) checkers, Type II checkers are error-input semi-sensitive (EISS) checkers, Type III checkers are error-input insensitive (EIIS) checkers, Type IV checkers are double-error-input sensitive (DEIS) checkers, and Type V checkers are conditional error-input sensitive (CEIS) checkers.

All five types of checkers can be used in designing various TSC checkers to achieve multiple error detection. But only Type I, Type II and Type III checkers are efficient for building various TSC functional circuits. Type IV and Type V chckers are very useful for constructing TSC checkers but their applications in TSC functional part remain unclear.

4.3 A group of strong morphic basic operations in B_{SM}

We have formalized a strong morphic Boolean algebra B_{SM} and a new classification of checkers. Here, we propose a group of strong morphic basic operations which implements three basic functions — XOR_{SM}, AND_{SM} and OR_{SM}. According to the new classification of checkers, we develop three types of basic operations for each basic function. For XOR_{SM} basic function, we define XOR_{SM1}, XOR_{SM2} and XOR_{SM3} operations; for AND_{SM}, we define AND_{SM1}, AND_{SM2} and AND_{SM3} operations; for OR_{SM}, we define OR_{SM1}, OR_{SM2}, and OR_{SM3} operations. These three types operations correspond to Type I, Type II, and Type III checkers, respectively.

4.3.1 Strong morphic operators

We define a strong morphic operator as:

Definition 12 : *An operator is a strong morphic operator if and only if it possesses the following two properties:*

- *It implements a natural correspondence between B_{SM} and B_u in the intrinsic state space;*
- *It functions as one of checkers among Type I, Type II and Type III checkers in extrinsic state space.*

We use the symbol $\{*_{PN}\}$ to stand for a group of morphic operators, $*$ for a natural correspondence between B_{SM} (or B_M) and B_u, and P for the type of morphic operator, i.e., when P is SM, it is a strong morphic operator, and it is a conventional morphic operator when P is M. N $\in \{1,2,3,4,5\}$, and it indicates to which type of checker does the operator belong. For example, $\oplus_{SM3}$ means that this operator is a strong morphic Type III XOR operator — a EIIS-XOR operator.

4.3.2 Strong morphic basic operations

We define a operation which involves only two variables as a basic operation. According to the definition of strong morphic operators, we propose three basic strong morphic functions — XOR, AND and OR — which give three natural correspondences between B_{SM} and B_u in the intrinsic state space. These natural correspondences are given in Table 4.1. They implement the following basic functions:

$$E = A \oplus_{SM} B \tag{4.24}$$

$$C = A \bullet_{SM} B \tag{4.25}$$

$$D = A +_{SM} B \tag{4.26}$$

Table 4.1: Three basic SM operations in intrinsic space.

Input A		Input B		AND_{SM} C		OR_{SM} D		XOR_{SM} E	
A_1	A_2	B_1	B_2	C_1	C_2	D_1	D_2	E_1	E_2
0	1	0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0	1	0
1	0	0	1	0	1	1	0	1	0
1	0	1	0	1	0	1	0	0	1

In the intrinsic state space, these three basic functions can implement any given combinational logic function.

For each basic function, we formalize three basic operations — $*_{SM1}$, $*_{SM2}$ and $*_{SM3}$. Thus, we have formalized nine strong morphic basic operations as

Group 1: XOR_{SM1}, AND_{SM1}, OR_{SM1};

Group 2: XOR_{SM2}, AND_{SM2}, OR_{SM2};

Group 3: XOR_{SM3}, AND_{SM3}, and OR_{SM3}.

In intrinsic state space, all these three groups implement the basic functions which are defined in Table 4.1. In extrinsic state space, they work like Type I, Type II, and Type III checkers, respectively. These are illustrated in Table 4.2, Table 4.3, and Table 4.4, respectively.

4.4 Concluding remarks

We have proposed a new Boolean algebra — strong morphic Boolean algebra B_{SM}. This new Boolean algebra overcomes the disadvantage of conventional morphic Boolean algebra B_M which is incapable of providing natural correspondence between B_M and B_u. While possessing all the properties of B_M, B_{SM} also provides

Table 4.2: The three basic SM1 operation in extrinsic space.

Input A	Input B	AND_{SM1} C	OR_{SM1} D	XOR_{SM1} E
(0 0)	(X X)	(00)/(11)	(00)/(11)	(00)/(11)
(1 1)	(X X)	(00)/(11)	(00)/(11)	(00)/(11)
(X X)	(0 0)	(00)/(11)	(00)/(11)	(00)/(11)
(X X)	(1 1)	(00)/(11)	(00)/(11)	(00)/(11)

Table 4.3: The three basic SM2 operation in extrinsic space.

Input A	Input B	AND_{SM2} C	OR_{SM2} D	XOR_{SM2} E
(01)/(10)	(00)/(11)	(00)/(11)	(00)/(11)	(00)/(11)
(00)/(11)	(01)/(10)	(00)/(11)	(00)/(11)	(00)/(11)
(00)/(11)	(00)/(11)	(01)/(10)	(01)/(10)	(01)/(10)

Table 4.4: The three basic SM3 operation in extrinsic space.

Input A	Input B	AND_{SM3} C	OR_{SM3} D	XOR_{SM3} E
(00)/(11)	(X X)	(01)/(10)	(01)/(10)	(01)/(10)
(X X)	(00)/(11)	(01)/(10)	(01)/(10)	(01)/(10)
(00)/(11)	(00)/(11)	(01)/(10)	(01)/(10)	(01)/(10)

a natural correspondence between B_{SM} and B_u. We have given a new classification for checkers. Five types of checkers have been defined. These five types of checkers formalize various behaviours of different checkers. Three strong morphic basic functions and corresponding nine basic operations have been formalized. This chapter provides a theoretical basis for developing universal TSC BBBs and designing TR-TSC circuits, especially for TSC functional circuits. Based on the theory presented, we will develop new groups of universal TR-TSC BBBs and provide design techniques of TR-TSC circuits using new TR-TSC BBBs.

Chapter 5

Two-rail totally self-checking basic building blocks

Based on strong morphic Boolean algebra and the new classification of checkers, three types of basic building blocks (BBB) have been developed in this chapter. These three types of BBBs are error-input sensitive BBBs, error-input semi-sensitive BBBs, and error-input insensitive BBBs. Each type includes three basic function blocks — AND BBB, OR BBB, and XOR BBB. Using these three basic function blocks one can implement any given combinational logic function. Properly using different types of BBBs, various TR-TSC circuits which have the capability of confining internal faults in separate areas can be achieved. In addition, we also present another set of universal BBBs called TR-TSC multi-function (MF) BBBs. The applications of BBBs will be described in *Chapter 6.*

5.1 TR-TSC error-input sensitive (EIS) basic building blocks

TR-TSC error-input sensitive (EIS) BBBs implement a group of extrinsic sensitive strong morphic Boolean basic operations — XOR_{SM1}, AND_{SM1}, and OR_{SM1}

— which have been defined in *Chapter 4*. TR-TSC EIS BBBs are composed of three TR-TSC BBBs — TR-TSC EIS-XOR BBB, TR-TSC EIS-AND BBB, and TR-TSC EIS-OR BBB. These BBBs are Type I checkers.

5.1.1 TR-TSC EIS-XOR basic building block

The proposed structure of TR-TSC EIS-XOR BBB[1] is shown in Figure 5.1. It is a conventional two-input two-variable comparator. Table 5.1 gives its truth table. It implements the following basic operation:

$$E = A \oplus_{SM1} B \tag{5.1}$$

Any single internal fault can be tested by the minimal testing set T_S:

$$T_S = \{T_1, T_2, T_3, T_4\} \tag{5.2}$$

where $T_1 = \{('01), (01)\}, T_2 = \{(01), (10)\}, T_3 = \{(10), (01)\}$, and $T_4 = \{('0), (10)\}$.

For example, (1). Suppose that a stuck-at-0 fault occurs at gate-2, its ouptut keeps correct value when its input vector is T_2 or T_3 or T_4; it indicates the fault by (00) when its input is T_1. (2). Suppose that there is a stuck-at-1 fault at gate-4, it keeps correct outputs when its inputs are T_2 and T_3; it will give the fault indication when it receives the input which is either T_1 or T_4 by (11). The verifications for the rest of stuck-at faults follow the similar procedures, and here are omitted.

Theorem 5 : *The proposed EIS-XOR BBB is a TR-TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in A.1 – A.6. It either keeps the correct value or gives an error indication at its output E when any

[1]Its internal construction is the same as SEDTR

Table 5.1: The truth table of TR-TSC EIS-XOR BBB.

Input A		Input B		Gates				Output E	
A_1	A_2	B_1	B_2	1	2	3	4	E_1	E_2
0	1	0	1	0	1	0	0	0	1
0	1	1	0	0	0	0	1	1	0
1	0	0	1	0	0	1	0	1	0
1	0	1	0	1	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0
0	1	1	1	0	1	0	1	1	1
0	0	0	1	0	0	0	0	0	0
1	1	0	1	0	1	1	0	1	1
1	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	1	0	1	1
0	0	1	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	1	1	1	1	1

single internal stuck-at fault occurs during normal operation; it possesses the *fault secure* property. Since any single internal stuck-at fault can be tested by at least one testing vector $T_i \in T_S$ when T_S is applied to its inputs, the *self-testing* property is also preserved. Consequently, the proposed EIS-XOR is a TR-TSC BBB. □

5.1.2 TR-TSC EIS-AND basic building block

The proposed structure of TR-TSC EIS-AND BBB is shown in Figure 5.2. The proposed EIS-AND BBB implements the following basic operation:

$$C = A \bullet_{SM1} B \tag{5.3}$$

Besides its main functional output C, it has another output E which implements

Table 5.2: The truth table of TR-TSC EIS-AND BBB.

Input A		Input B		Output E		Output C	
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1
1	0	1	0	0	1	1	0
0	1	0	0	0	0	0	0
0	1	1	1	1	1	0	0
0	0	0	1	0	0	0	0
1	1	0	1	1	1	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1
0	0	1	0	0	0	0	0
1	1	1	0	1	1	1	1
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0

XOR_{SM1} operation. It is an EIS-XOR function output.

Its truth table is given in Table 5.2.

Theorem 6 : *The proposed EIS-AND BBB is a TR-TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in A.7 – A.16. The rest of the proof is similar to in Theorem 5. Consequently, the proposed EIS-AND is a TR-TSC BBB. □

5.1.3 TR-TSC EIS-OR basic building block

The proposed structure of TR-TSC EIS-OR BBB is shown in Figure 5.3. The proposed EIS-OR BBB implements the following basic operation:

$$D = A +_{SM1} B \tag{5.4}$$

Besides its main functional output D, it has another output E which implements XOR_{SM1} operation. It is an EIS-XOR function output.

Its truth table is given in Table 5.3.

Theorem 7 : *The proposed EIS-OR BBB is a TR-TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in A.17 - A.26. The rest of the proof is similar to in Theorem 5. Consequently, the proposed EIS-OR is a TR-TSC BBB. □

5.2 TR-TSC error-input semi-sensitive (EISS) basic building blocks

TR-TSC error-input semi-sensitive (EISS) BBBs implement a group of extrinsic semi-sensitive strong morphic Boolean basic operations — XOR_{SM2}, AND_{SM2}, and OR_{SM2} — which have been defined in *Chapter 4*. TR-TSC EISS BBBs are composed of three TR-TSC BBBs — TR-TSC EISS-XOR BBB, TR-TSC EISS-AND BBB, and TR-TSC EISS-OR BBB. These BBBs are Type II checkers.

Table 5.3: The truth table of TR-TSC EIS-OR BBB.

Input A		Input B		Output E		Output D	
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1
0	1	1	0	1	0	1	0
1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0
0	1	0	0	0	0	0	0
0	1	1	1	1	1	1	1
0	0	0	1	0	0	0	0
1	1	0	1	1	1	1	1
1	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0
0	0	1	0	0	0	0	0
1	1	1	0	1	1	0	0
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0

5.2.1 TR-TSC EISS-XOR basic building block

The proposed structure of TR-TSC EISS-XOR BBB is shown in Figure 5.4. The proposed EISS-XOR BBB implements the following basic operation:

$$E = A \oplus_{SM2} B \tag{5.5}$$

Its truth table is given in Table 5.4.

Theorem 8 : *The proposed EISS-XOR BBB is a TR-TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in B.1 and B.2. The rest of the proof is similar to in Theorem 5. Consequently, the proposed EISS-XOR is a TR-TSC BBB. □

Table 5.4: The truth table of TR-TSC EISS-XOR BBB.

Input A		Input B		Output E	
A_1	A_2	B_1	B_2	E_1	E_2
0	1	0	1	0	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	0	1
0	1	0	0	0	0
0	1	1	1	1	1
0	0	0	1	0	0
1	1	0	1	1	1
1	0	0	0	1	1
1	0	1	1	0	0
0	0	1	0	1	1
1	1	1	0	0	0
0	0	0	0	0	1
0	0	1	1	1	0
1	1	0	0	1	0
1	1	1	1	0	1

5.2.2 TR-TSC EISS-AND basic building block

The proposed structure of TR-TSC EISS-AND BBB is shown in Fig 5.5. The proposed EISS-AND BBB implements the following basic operation:

$$C = A \bullet_{SM2} B \tag{5.6}$$

Besides its main functional output C, it has another output E which implements XOR_{SM2} operation. It is an EISS-XOR function output.

Its truth table is given in Table 5.5.

Theorem 9 : *The proposed EISS-AND BBB is a TR-TSC BBB.*

Table 5.5: The truth table of TR-TSC EISS-AND BBB.

Input **A**		Input **B**		Output **E**		Output **C**	
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1
1	0	1	0	0	1	1	0
0	1	0	0	1	1	1	1
0	1	1	1	0	0	1	1
0	0	0	1	1	1	1	1
1	1	0	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1
0	0	1	0	0	0	0	0
1	1	1	0	1	1	1	1
0	0	0	0	0	1	1	0
0	0	1	1	1	0	0	1
1	1	0	0	1	0	0	1
1	1	1	1	0	1	0	1

Proof: Its *fault secure* and *self-testing* properties are illustrated in B.3 – B.8. The rest of the proof is similar to in Theorem 5. Consequently, the proposed EISS-AND is a TR-TSC BBB. □

5.2.3 TR-TSC EISS-OR basic building block

The proposed structure of TR-TSC EISS-OR BBB is shown in Figure 5.6. The proposed EISS-OR BBB implements the following basic operation:

$$D = A +_{SM2} B \tag{5.7}$$

Besides its main functional output D, it has another output E which implements XOR_{SM2} operation. It is an EISS-XOR function output.

Table 5.6: The truth table of TR-TSC EISS-OR BBB.

Input **A**		Input **B**		Output **E**		Output **D**	
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1
0	1	1	0	1	0	1	0
1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0
0	1	0	0	0	0	0	0
0	1	1	1	1	1	1	1
0	0	0	1	0	0	0	0
1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	1	0	0	1	1
0	0	1	0	1	1	1	1
1	1	1	0	0	0	1	1
0	0	0	0	0	1	0	1
0	0	1	1	1	0	1	0
1	1	0	0	1	0	1	0
1	1	1	1	0	1	1	0

Its truth table is given in Table 5.6.

Theorem 10 : *The proposed EISS-OR BBB is a TR-TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in B.9 - B.14. The rest of the proof is similar to in Theorem 5. Consequently, the proposed EISS-OR is a TR-TSC BBB. □

5.3 TR-TSC error-input insensitive (EIIS) basic building blocks

TR-TSC error-input insensitive (EIIS) BBBs implement a group of extrinsic insensitive strong morphic Boolean basic operations — XOR_{SM3}, AND_{SM3}, and OR_{SM3} — which have been defined in *Chapter 4*. TR-TSC EIIS BBBs are composed of three TR-TSC BBBs — TR-TSC EIIS-XOR BBB, TR-TSC EIIS-AND BBB, and TR-TSC EIIS-OR BBB. These BBBs are Type III checkers.

5.3.1 TR-TSC EIIS-XOR basic building block

Proposed structure of TR-TSC EIIS-XOR BBB is shown in Figure 5.7. The proposed EIIS-XOR BBB implements the following basic operation:

$$E^* = A \oplus_{SM3} B \tag{5.8}$$

Besides its main functional output E^*, it has three other outputs — E, C, and D. E is an EISS-XOR function output which implements XOR_{SM2} operation; C is an EISS-AND function output which implements AND_{SM2} operation; D is an EIIS-OR function output which implements OR_{SM3} operation.

Its truth table is given in Table 5.7.

Theorem 11 : *The proposed EIIS-XOR BBB is a TR-TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in C.1 – C.26. The rest of the proof is similar to in Theorem 5. Consequently, the proposed EIIS-XOR is a TR-TSC BBB. □

Table 5.7: The truth table of TR-TSC EIIS-XOR BBB.

Input A		Input B		Output E		Output C		Output D		Output E^*	
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0	1	0	0	1
0	1	0	0	1	1	1	1	1	0	1	0
0	1	1	1	0	0	1	1	0	1	0	1
0	0	0	1	1	1	1	1	1	0	1	0
1	1	0	1	0	0	1	1	0	1	0	1
1	0	0	0	0	0	0	0	1	0	1	0
1	0	1	1	1	1	1	1	1	0	0	1
0	0	1	0	0	0	0	0	1	0	1	0
1	1	1	0	1	1	1	1	1	0	0	1
0	0	0	0	0	1	1	0	1	0	1	0
0	0	1	1	1	0	0	1	1	0	1	0
1	1	0	0	1	0	0	1	1	0	1	0
1	1	1	1	0	1	0	1	0	1	1	0

When T_S is applied to its inputs, every single internal stuck-at fault is reflected at its main output E^* for at least one testing vector $T_i \in T_S$ except the single stuck-at faults at gate 1 and gate 2 which are indicated at E and C.

5.3.2 TR-TSC EIIS-AND basic building block

The proposed structure of TR-TSC EIIS-AND BBB is shown in Figure 5.8. The proposed EIIS-AND BBB implements the following basic operation:

$$C = A \bullet_{SM3} B \tag{5.9}$$

Besides its main functional output C, it has two other outputs — E and D. E is an EISS-XOR function output which implements XOR_{SM2} operation; D is an

Table 5.8: The truth table of TR-TSC EIIS-AND BBB.

Input A		Input B		Output E		Output D		Output C	
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	C_1	C_2
0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	1	0	0	1
1	0	0	1	1	0	1	0	0	1
1	0	1	0	0	1	1	0	1	0
0	1	0	0	0	0	0	0	0	1
0	1	1	1	1	1	1	1	0	1
0	0	0	1	0	0	0	0	0	1
1	1	0	1	1	1	1	1	0	1
1	0	0	0	1	1	1	1	0	1
1	0	1	1	0	0	1	1	1	0
0	0	1	0	1	1	1	1	0	1
1	1	1	0	0	0	1	1	1	0
0	0	0	0	0	1	0	1	0	1
0	0	1	1	1	0	1	0	0	1
1	1	0	0	1	0	1	0	0	1
1	1	1	1	0	1	1	0	1	0

EISS-OR function output which implements OR_{SM2} operation.

Its truth table is given in Table 5.8.

Theorem 12 : *The proposed EIIS-AND BBB is a TR-TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in C.27 - C.35. The rest of the proof is similar to in Theorem 5. Consequently, the proposed EIIS-AND is a TR-TSC BBB. □

When T_S is applied to its inputs, every single internal stuck-at fault is reflected at its main output C for at least one testing vector $T_i \in T_S$ except the single stuck-at faults at gate 1 and gate 2 which are indicated at E.

Table 5.9: The truth table of TR-TSC EIIS-OR BBB.

Input A		Input B		Output E		Output C		Output D	
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0
1	0	0	1	1	0	0	1	1	0
1	0	1	0	0	1	1	0	1	0
0	1	0	0	1	1	1	1	1	0
0	1	1	1	0	0	1	1	0	1
0	0	0	1	1	1	1	1	1	0
1	1	0	1	0	0	1	1	0	1
1	0	0	0	0	0	0	0	1	0
1	0	1	1	1	1	1	1	1	0
0	0	1	0	0	0	0	0	1	0
1	1	1	0	1	1	1	1	1	0
0	0	0	0	0	1	1	0	1	0
0	0	1	1	1	0	0	1	1	0
1	1	0	0	1	0	0	1	1	0
1	1	1	1	0	1	0	1	0	1

5.3.3 TR-TSC EIIS-OR basic building block

Proposed structure of TR-TSC EIIS-OR BBB is shown in Figure 5.9. The proposed EIIS-OR BBB implements the following basic operation:

$$D = A +_{SM3} B \tag{5.10}$$

Besides its main functional output D, it has two other outputs — E and C. E is an EISS-XOR function output which implements XOR_{SM2} operation; C is an EISS-AND function output which implements AND_{SM2} operation.

Its truth table is given in Table 5.9.

Theorem 13 : *The proposed EIIS-OR BBB is a TR-TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in C.36 – C.44. The rest of the proof is similar to in Theorem 5. Consequently, the proposed EIIS-OR is a TR-TSC BBB. □

When T_S is applied to its inputs, every single internal stuck-at fault is reflected at its main output D for at least one testing vector $T_i \in T_S$ except the single stuck-at faults at gate 1 and gate 2 which are indicated at E.

5.4 TR-TSC Multi-function (MF) basic building blocks

A universal set of TR-TSC MF BBBs [43] is proposed here . This set of BBBs implements the functions of the morphic basic operations formalized by Gaitanis [11]–[16] and some strong morphic basic operations described in *Chapter 4*. It consists of a TR-TSC MF-OR BBB, a TR-TSC MF-AND BBB, and a TR-TSC MF-XOR BBB. TR-TSC MF-OR BBB incorporates the other two MF BBBs – MF-AND BBB and MF-XOR BBB. It is a multi-type checker which includes Type I, Type II, Type IV, and Type V functions. TR-TSC MF-AND BBB is a Type I checker, and TR-TSC MF-XOR BBB is a Type II checker.

5.4.1 TR-TSC MF-OR basic building block

TR-TSC MF-OR BBB is multi-function BBB. It provides all the functions of the morphic basic operations formalized by Gaitanis [11]–[16] and some strong morphic basic operations described in *Chapter 4*. These functions are:

$$D^* = A +_{SM1} B \tag{5.11}$$

$$D = A +_{M4} B \tag{5.12}$$

Table 5.10: The truth table of TR-TSC MF-OR BBB.

Input A		Input B		MAND C''		MOR D'		MCOR F'		MLXOR E'		AND C'		OR D		EXOR E		OR D'	
A_1	A_2	B_1	B_2	C''_1	C''_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_1	E_2	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	1	0
1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	1	1
1	0	1	1	1	1	0	1	1	1	0	0	1	1	0	1	1	1	1	1
0	1	0	0	0	0	1	0	0	0	1	1	0	0	1	0	1	1	0	0
0	1	1	1	1	1	1	0	0	0	1	1	0	0	0	1	0	0	1	1
0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1	1
1	1	1	0	1	1	1	0	1	0	1	1	1	1	0	1	1	1	1	1
0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	1	1	1	1
1	1	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	1	0	1	1	0	0	0	1	1	1	0	1	1
0	0	1	1	0	0	1	1	0	1	1	0	0	0	1	1	0	1	1	1
1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0
1	1	1	1	1	1	0	0	1	0	0	1	0	0	1	1	1	0	1	1

$$D' = A +_{M3} B \tag{5.13}$$

$$C = A \bullet_{SM1} B \tag{5.14}$$

$$C' = A \bullet_{M1} B \tag{5.15}$$

$$E = A \oplus_{SM2} B \tag{5.16}$$

$$E' = \overline{A \oplus_{M2} B} \tag{5.17}$$

$$F' = A +_{M5} B \tag{5.18}$$

The internal construction of TR-TSC MF-OR BBB is shown in Figure 5.10 and its truth table is given in Table 5.10.

Theorem 14 : *The proposed TR MF-OR BBB is a TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in D.1 – D.52. The rest of the proof is similar to in Theorem 5. Consequently, the proposed TR MF-OR BBB is a TSC BBB. □

For example, assume that there is a stuck-at-1 fault at the output of gate 17 in type 2 block. This fault can be tested by T_2, $T_3 \in T_s$, and will be indicated at its output D^* by (00) (the outputs — D' and D also indicate this fault). If the output of gate 2 has a stuck-at-0 fault in type 2 block, this fault can be tested by T_1 and only indicated at its output C' by (00).

When T_S is applied to its inputs, every single internal stuck-at fault is reflected at its main output D' for at least one testing vector $T_i \in T_S$ except the single stuck-at-0 fault at gate 2 which is only indicated at C'. Thus, the output C' also has to be monitored when using TR-TSC MF-OR BBBs to building TR-TSC circuits.

5.4.2 TR-TSC MF-AND basic building block

The internal construction of TR-TSC MF-AND BBB which implements

$$C = A \bullet_{SM1} B, \tag{5.19}$$

$$C' = \overline{A \oplus_{SM1} B} \tag{5.20}$$

is shown in Figure 5.11.

Theorem 15 : *The proposed TR MF-AND BBB is a TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in D.1 - D.23. The rest of the proof is similar to in Theorem 5. Consequently, the proposed TR MF-AND BBB is a TSC BBB. □

For example, assume that the output of gate 4 has a stuck-at-0 fault, this fault can be tested by T_1. The outputs C' and C indicate this fault by (00) and (11) respectively.

5.4.3 TR-TSC MF-XOR basic building block

The internal construction of TR-TSC MF-XOR BBB which implements

$$E = A \oplus_{SM2} B, \tag{5.21}$$

$$E' = \overline{A \oplus_{M2} B} \tag{5.22}$$

is shown in Figure 5.12. $\oplus_{M2}$ is the odd-parity operation [11]-[16].

Theorem 16 : *The proposed TR MF-XOR BBB is a TSC BBB.*

Proof: Its *fault secure* and *self-testing* properties are illustrated in D.40 – D.48. The rest of the proof is similar to in Theorem 5. Consequently, the proposed TR MF-XOR BBB is a TSC BBB. □

For example, assume that the output of gate 2 has a stuck-at-1 fault; this fault can be tested by $T_1, T_2 \in T_s$ and will be indicated at its output E by (00) and (11) respectively.

5.5 Comparisons of four sets of TR-TSC basic building blocks

Table 5.11 lists the number of logic gates used in different BBBs. It also gives the gate levels in the BBBs.

In comparison with TR-TSC EIS BBBs and TR-TSC EIIS BBBs, TR-TSC EIS BBBs use less hardware and has a lewer gate levels; howevere, they are not widely used in designing TR-TSC circuits. TR-TSC EIS BBBs and TR-TSC EIIS BBBs have acceptable number of gates and gate levels. They are the most useful BBBs in building TR-TSC functional circuits while they also can be applied to the design

Table 5.11: The table of comparisons of different BBBs.

TYPE OF BLOCKS	AND/OR	XOR/$\overline{XOR}$	Inverter	Total Gates	Gate Levels
EIS-XOR	6	0	0	6	2
EIS-AND	8	2	0	10	3
EIS-OR	8	2	0	10	3
EISS-XOR	0	2	0	2	1
EISS-AND	2	4	0	6	2
EISS-OR	2	4	0	6	2
EIIS-XOR	4	8	1	13	5
EIIS-AND	2	6	1	9	4
EIIS-OR	2	6	1	9	4
EXOR BBB	0	4	0	4	2
AND BBB	9	2	0	11	4
OR BBB	12	14	0	26	6

of TSC checkers. TR-TSC MF BBBs possess the functions of both the morphic basic operations proposed by Gaitanis [11]–[16] and some strong morphic basic operations described in *Chapter 4*; therefore, they cost much hardware and have more gate levels. Thus, they are only suitable for cases which have modular cell structure.

5.6 Concluding remarks

We have proposed four sets of universal TR-TSC BBBs — TR-TSC EIS BBBs, TR-TSC EISS BBBs, TR-TSC EIIS BBBs, and TR-TSC MF BBBs. These BBBs can be used in building TR-TSC circuits which implement any given combinational logic function. Since EIS BBBs and EIIS BBBs possess useful properties, they are essential for designing TR-TSC functional circuits. This will be discussed in *Chapter 6*.

When EIIS BBBs are used in building TR-TSC functional circuits, both main function outputs and output E have to be monitored because single stuck-at faults at gate 1 and gate 2 are only reflected at the output E.

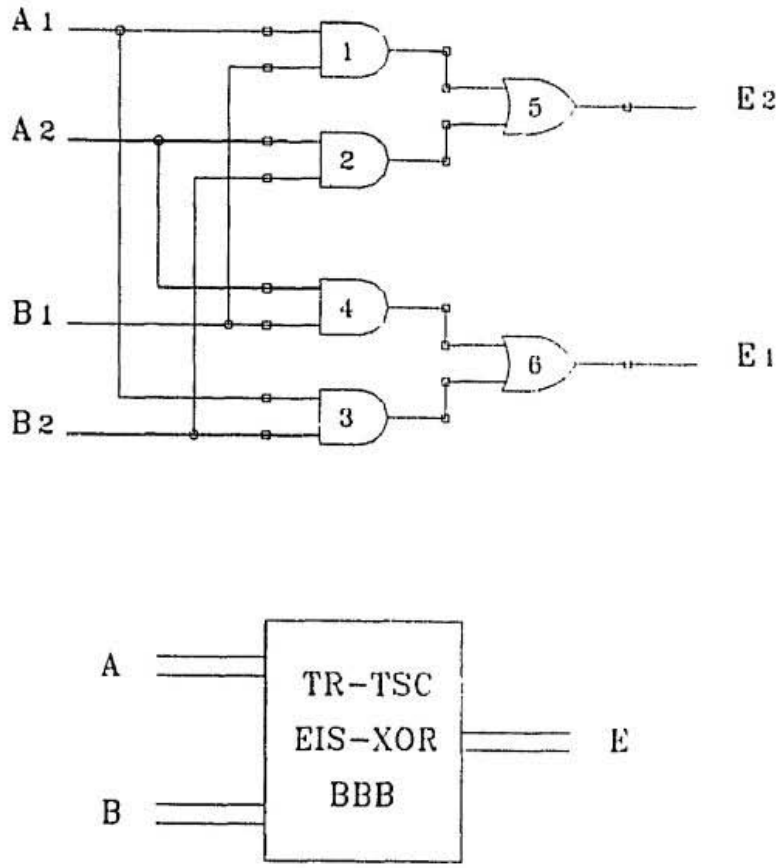

Figure 5.1: Proposed structure of TR-TSC EIS-XOR BBB.

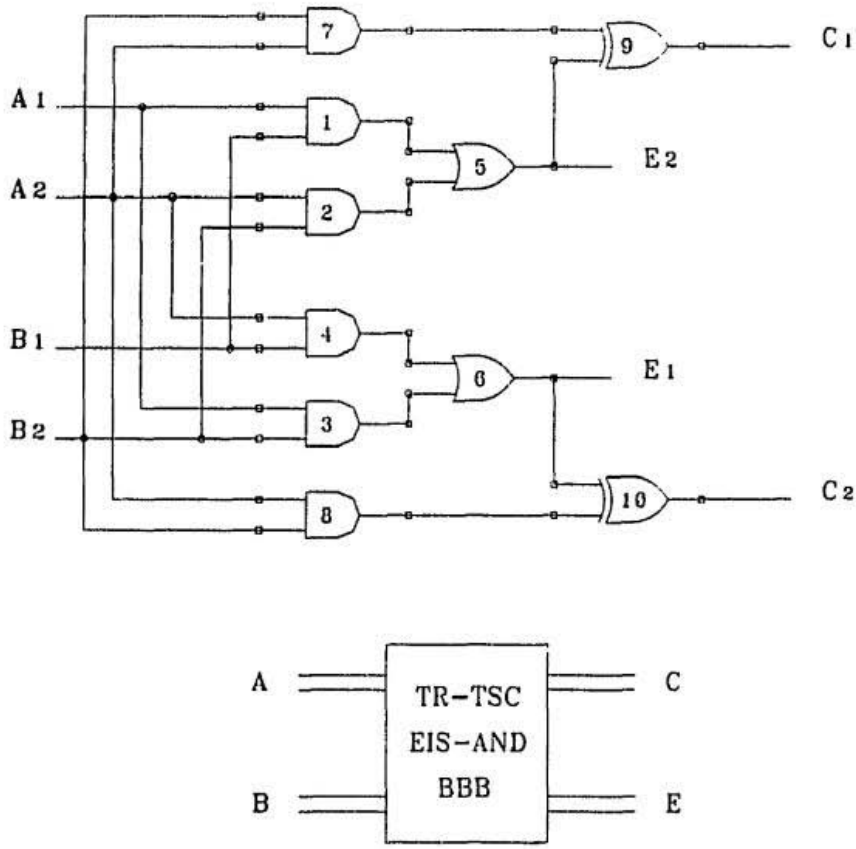

Figure 5.2: Proposed structure of TR-TSC EIS-AND BBB.

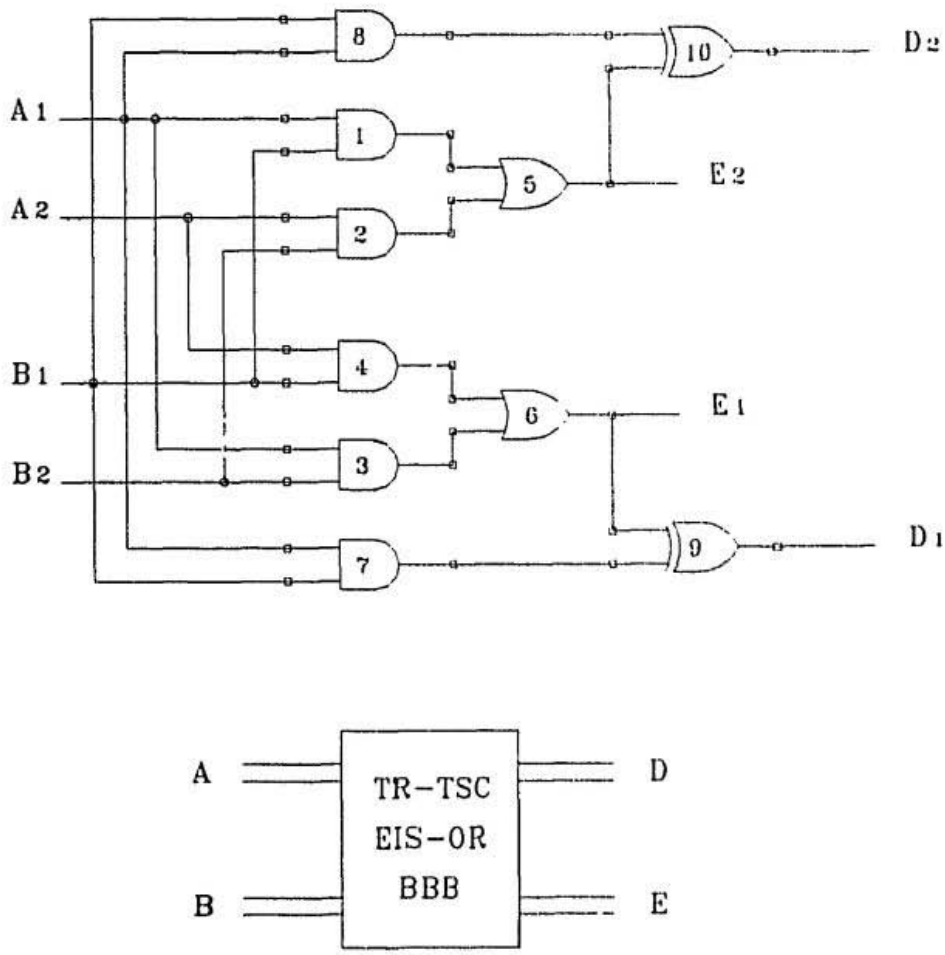

Figure 5.3: Proposed structure of TR-TSC EIS-OR BBB.

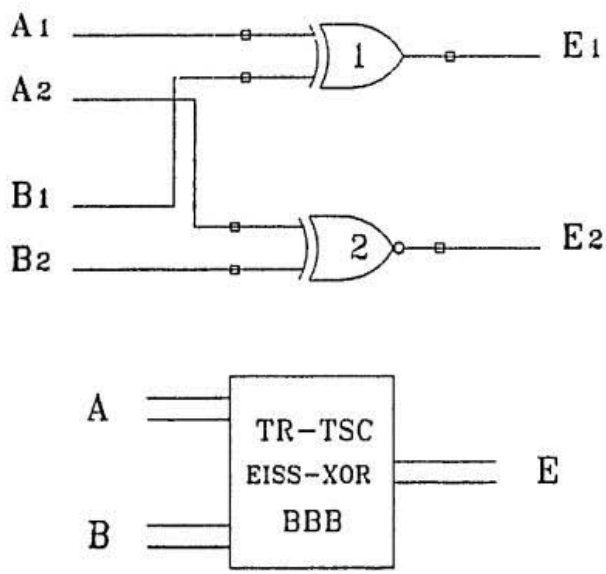

Figure 5.4: Proposed structure of TR-TSC EISS-XOR BBB.

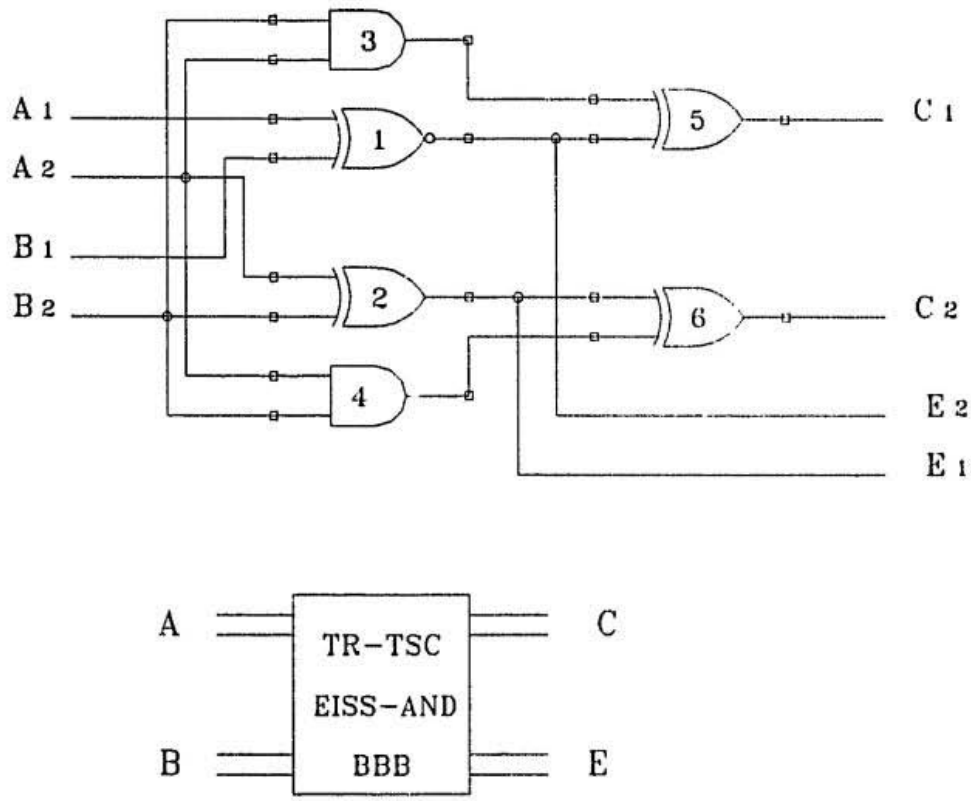

Figure 5.5: Proposed structure of TR-TSC EISS-AND BBB.

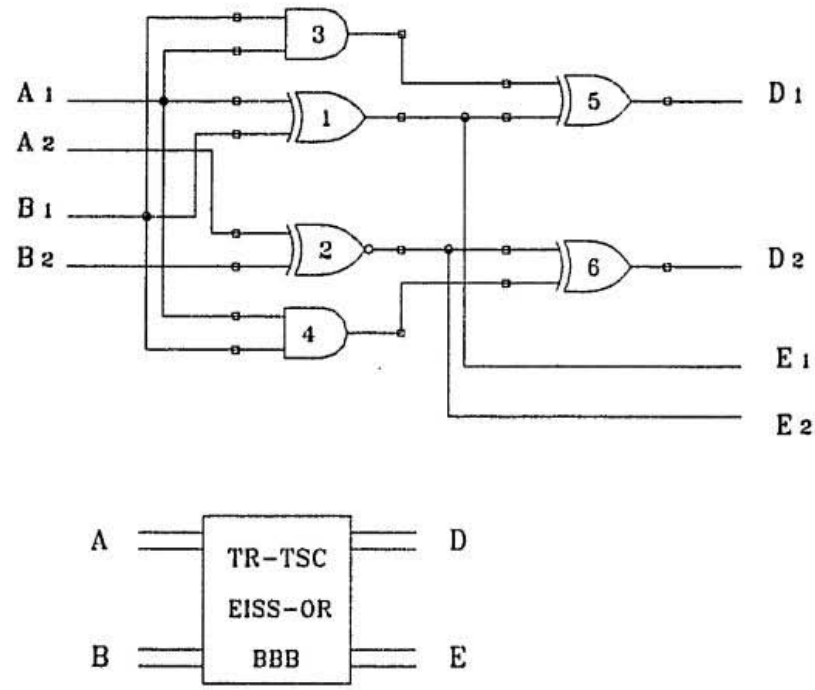

Figure 5.6: Proposed structure of TR-TSC EISS-OR BBB.

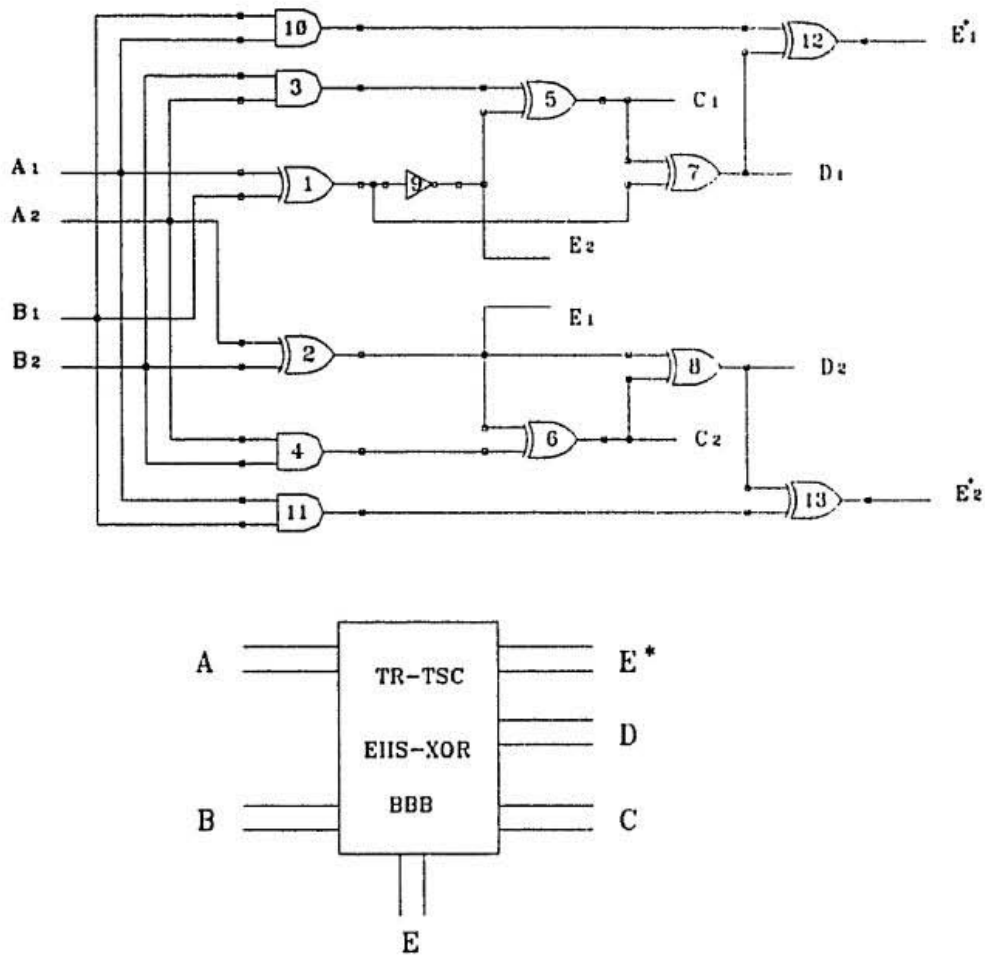

Figure 5.7: Proposed structure of TR-TSC EIIS-XOR BBB.

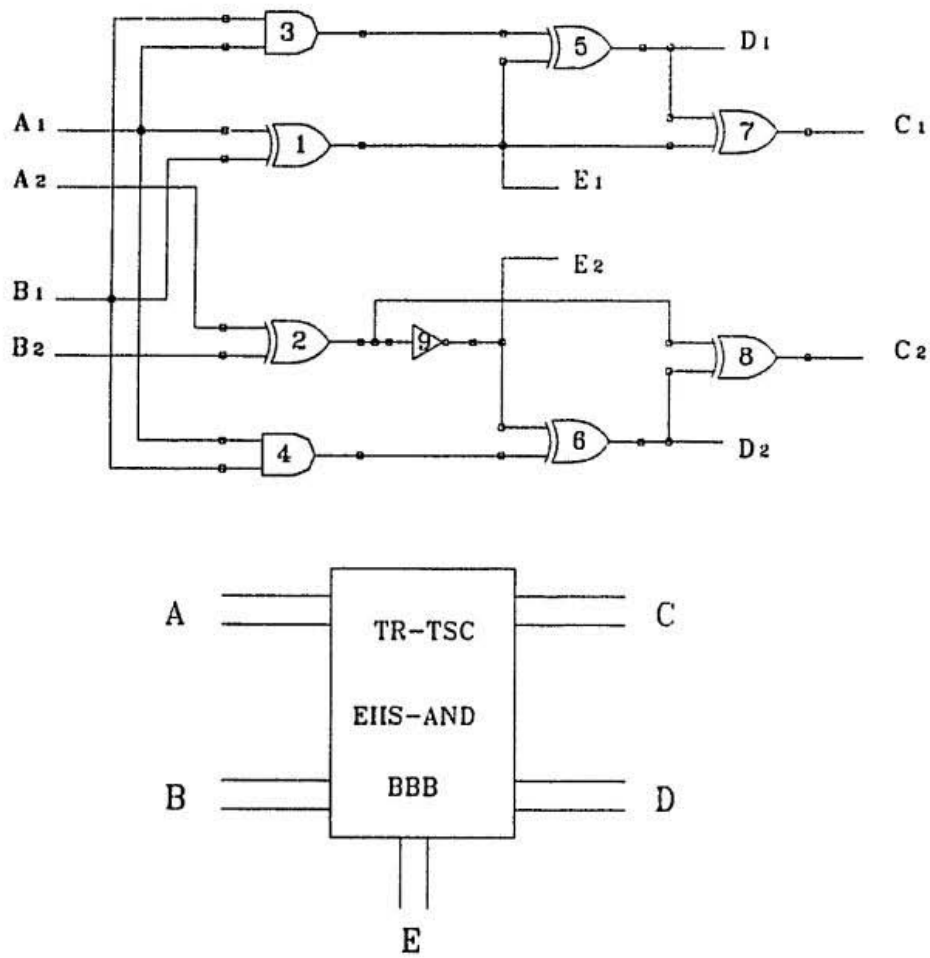

Figure 5.8: Proposed structure of TR-TSC EIIS-AND BBB.

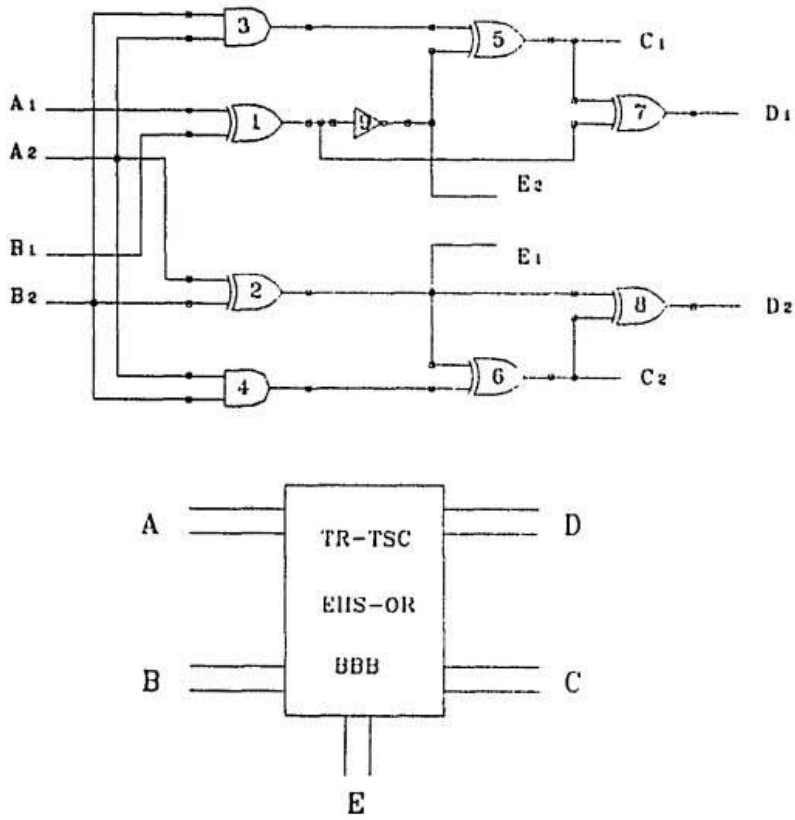

Figure 5.9: A proposed structure of TR-TSC EIIS-OR BBB.

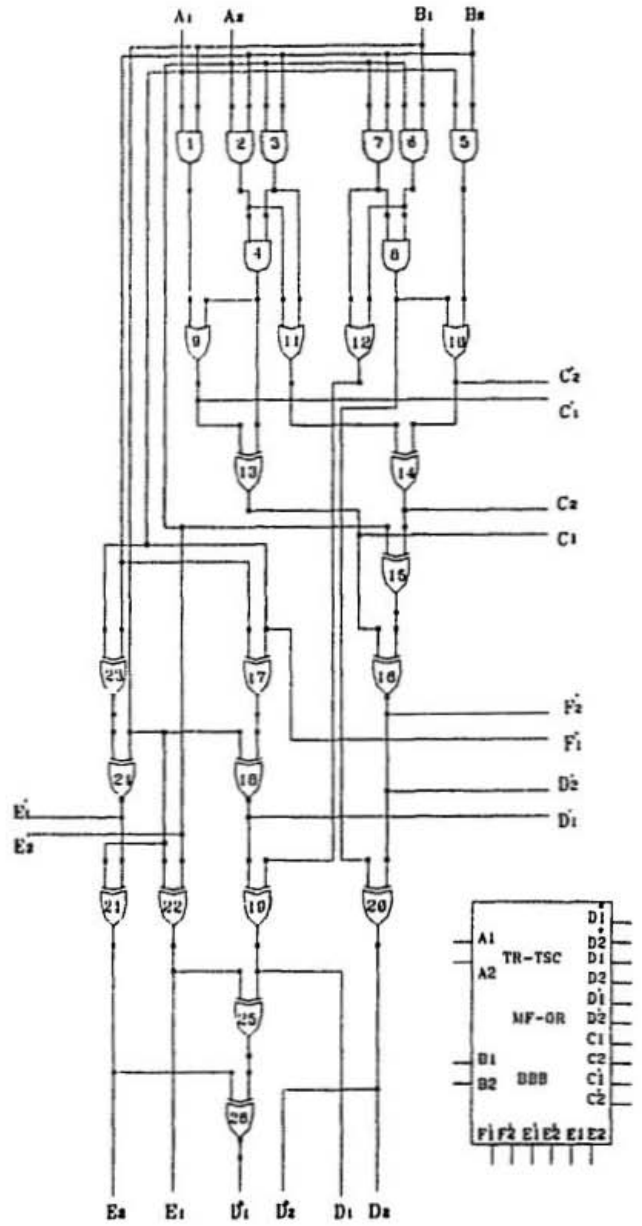

Figure 5.10: Proposed structure of TR-TSC MF-OR BBB.

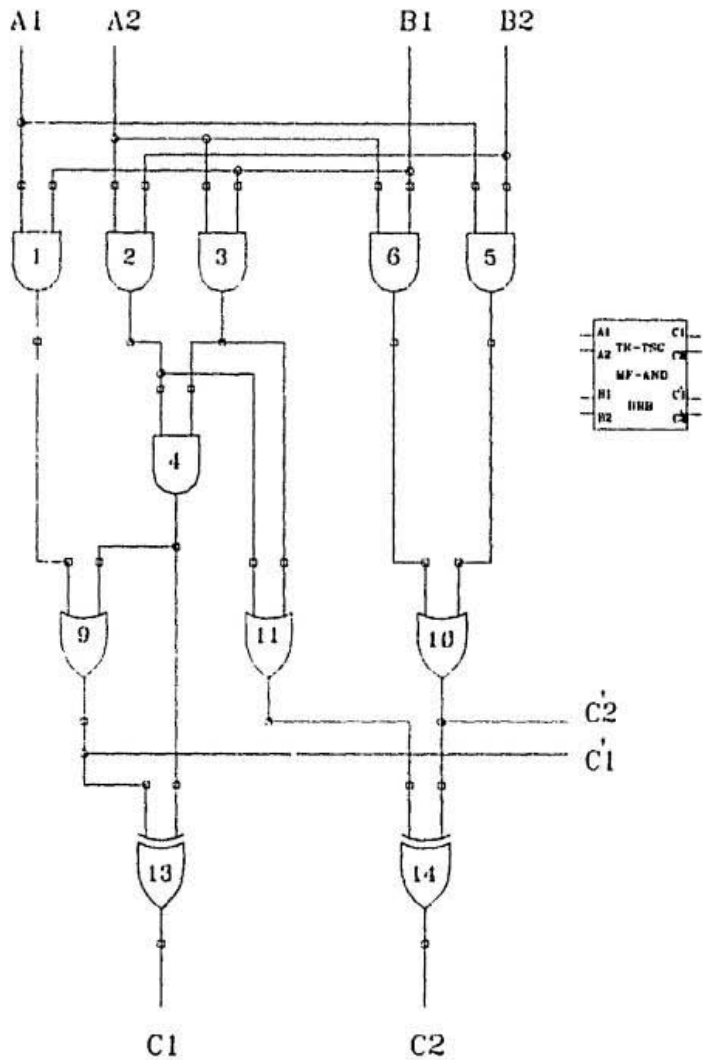

Figure 5.11: A proposed structure of TR-TSC MF-AND BBB.

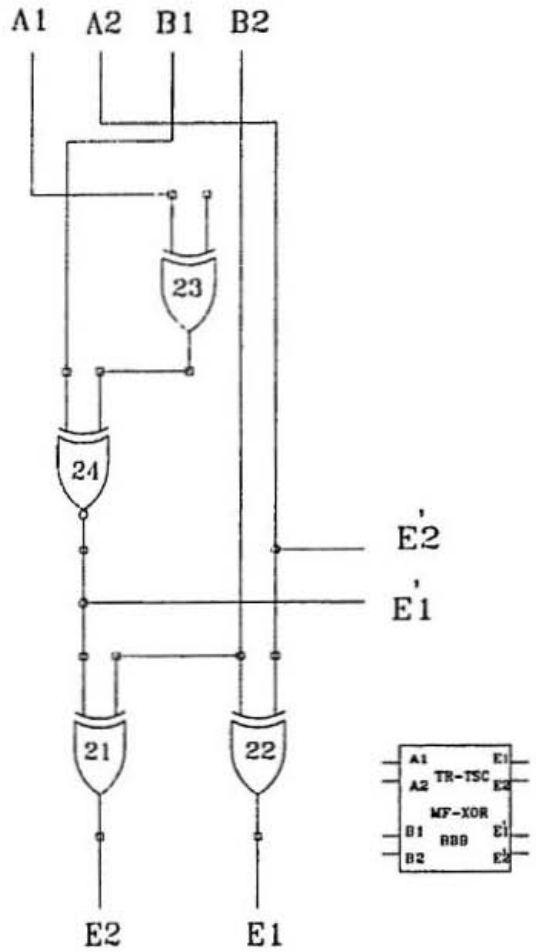

Figure 5.12: Proposed structure of TR-TSC MF-XOR BBB.

Chapter 6

Design of two-rail totally self-checking functional circuits

It is highly desirable that digital circuits possess TSC property. Since the TSC concept was introduced by Carter *et al* in 1968, a lot of work has been done on the design of TSC circuits. However, there is still no simple, convenient, flexible, and systematic design method available for TSC circuits, especially for TSC functional circuits. In many cases, to achieve a TSC circuit is very difficult and even impossible. In addition, the structure of conventional TSC circuits is usually complicated and makes it very difficult to locate faulty units. This leads to a low maintainability.

In order to overcome the limitations of conventional theories and design methods for TSC circuits, new design theories and methods are formalized in this chapter. New design methods are based on B_{SM} and the new class of checkers, and make use of the proposed new BBBs for constructing TR-TSC functional circuits. We also use existing design techniques of TR-TSC checkers to build TR-TSC error propagation circuits.

6.1 Simple interconnection method (SIM)

A design method called simple interconnection method (SIM) using new BBBs is presented here. SIM is suitable for the case that the *self-testing* property can be easily achieved or verified and has a large amount of inputs.

A functional circuit which is composed of the proposed BBBs and/or the primitive operator blocks [11]-[16] is *a TSC circuit* if and only if it meets the following conditions:

Condition 1 : *All the basic building blocks in the functional circuit receive the complete minimal test set T_S in their testing vectors under normal condition when a set of diagnostic sequence pairs is applied to the inputs of the circuit.*

Condition 2 : *There exists at least one path which makes every single internal fault propagate to one (or a group) of observable error indication output(s).*

Condition 3 : *All the components which form the error propagation circuits are able to receive the complete minimal test set T_S in their testing vectors during normal operation.*

The structure of this new class of TSC functional circuits consists of two parts. One part implements given combinational logic functions, the other part provides the path to propagate error indications. The new BBBs are highly suitable for the first part while they can also be used in constructing the second part, and the primitive operator blocks are efficient for the second part.

We give an example to demonstrate SIM for the design of TSC functional circuits using the proposed BBBs and the primitive operator blocks.

6.1.1 The Design of TR-TSC full (FA) adder using TR-TSC MF basic building blocks

Full adder implements the following logic functions:

$$S_i = X_i \oplus Y_i \oplus C_{i-1} \tag{6.1}$$

$$C_i = X_i \cdot Y_i + X_i \cdot C_{i-1} + Y_i \cdot C_{i-1} \tag{6.2}$$

where S_i – sum, C_i – carry, X_i, Y_i – inputs, C_{i-1} – previous carry.

A structure of TR-TSC full adder is shown in Figure 6.1.

Any single internal fault will propagate to an observable error indication output IF and any error input will be indicated at output EI. This TR-TSC full adder implementation requires considerably more gates than the TR-TSC full adder circuit in Figure 4.1. But, the method presented here is general for any function.

Theorem 17 : *The full adder described above is a TSC full adder.*

Proof: The proposed full adder maps code inputs into code outputs and noncode inputs into noncode outputs during fault-free operation. Thus, it is *code-disjoint.* Since the adder either keeps the correct value or gives at least one error indication at its error indication outputs EI and IF when any error inputs or any single internal stuck-at fault occurs during normal operation, it possesses the *fault secure* property. Its *self-testing* property can be verified by a set of 6-bit diagnostic sequence pairs W_i. The complete set of 6-bit diagnostic sequence pairs are given in E.1 — E.5. With the 6-bit diagnostic sequence pairs, it has been verified that each component is able to receive T_S in its testing vectors during normal operation, which is illustrated in Figure 6.1. Consequently, the full adder is a TSC full adder. □

Since this TSC full adder uses two big BBBs and many other BBBs (SEDTRs), it is not an economic scheme. Later, we will present a better scheme using EIS BBBs and SEDTRs.

6.1.2 Decoupling techniques for relevant error indication variables

We propose a decoupling method to achieve separate error indications of EI and IF in this section. This will help to locate faulty units and enhances maintainability.

A TSC decoupling BBB (DC_2)

The proposed TSC FA in Section 6.1.1 has two error indication outputs – EI and IF. However, IF is not independent of the inputs of the adder. When any error input appears, both EI and IF indicate this fault by (00) or (11). In this case, C_i and S_i have to be disregarded; therefore, it will be desirable to have a separate IF.

In order to solve the above problem, we propose a special BBB called TR-TSC decoupling BBB (DC_2). Here, we develop two types of DC_2. The Type I DC_2 employs less hardware than the Type II DC_2 but its internal faults are reflected at both its outputs EI and IF. In comparison to the Type I DC_2, any single stuck-at fault in the Type II DC_2 can be indicated at its output IF.

The internal constructions of Type I DC_2 and Type II DC_2 are shown in Figure 6.2 and Figure 6.3, respectively. The corresponding truth table is given in Table 6.1.

From Table 6.1, we find that there is no case which both EI and IF are (00) or (11). If this case occurs, it means that there is a single stuck-at fault in DC_2.

Theorem 18 : *The proposed Type I DC_2 is a TSC BBB.*

Proof: It is similar to the proof of **Theorem 17**. Here, it will not be repeated.

Similarly, the Type II DC_2 is also a TSC BBB.

A decoupling technique for relevant error indication variables

We give an example to demonstrate the decoupling method of relevant error indication variables.

Figure 6.4 is a structure of a TR-TSC decoupling checker (DC_4) with four relevant error indication variables and its truth table is given in Table 6.2.

Theorem 19 : *The proposed DC_4 is a TSC checker.*

Proof: It is similar to the proof of **Theorem 17**. Here, it will not be repeated.

An example application of DC_4 is shown in Figure 6.5. Its internal fault detection is illustrated in Table 6.3.

$$D = (((X0 \cdot X1) \oplus X2) \cdot X3) + X4 \tag{6.3}$$

The relationships of four variables are:

$$A \longrightarrow B \longrightarrow C \longrightarrow D \tag{6.4}$$

Table 6.1: The truth table of DC_2.

EI1	IF1	EI	IF
(01)/(10)	(01)/(10)	(01)/(10)	(01)/(10)
(00)/(11)	(00)/(11)	(00)/(11)	(01)/(10)
(01)/(10)	(00)/(11)	(01)/(10)	(00)/(11)
(00)/(11)	(01)/(10)	(00)/(11)	(01)/(10)

Table 6.2: The truth table of DC_4.

A	B	C	D	A_4	B_4	C_4	D_4
(00)/(11)	(00)/(11)	(00)/(11)	(00)/(11)	(00)/(11)	(01)/(10)	(01)/(10)	(01)/(10)
(01)/(10)	(00)/(11)	(00)/(11)	(00)/(11)	(01)/(10)	(00)/(11)	(01)/(10)	(01)/(10)
(01)/(10)	(01)/(10)	(00)/(11)	(00)/(11)	(01)/(10)	(01)/(10)	(00)/(11)	(01)/(10)
(01)/(10)	(01)/(10)	(01)/(10)	(00)/(11)	(01)/(10)	(01)/(10)	(01)/(10)	(00)/(11)

Table 6.3: An example of internal fault detection using DC_4.

Stuck at Fault	A	B	C	D	IF_1	IF_2	IF_3	IF_4
Q_1	(00)/(11)	(00)/(11)	(00)/(11)	(00)/(11)	(00)/(11)	(01)/(10)	(01)/(10)	(01)/(10)
Q_2	(01)/(10)	(00)/(11)	(00)/(11)	(00)/(11)	(01)/(10)	(00)/(11)	(01)/(10)	(01)/(10)
Q_3	(01)/(10)	(01)/(10)	(00)/(11)	(00)/(11)	(01)/(10)	(01)/(10)	(00)/(11)	(01)/(10)
Q_4	(01)/(10)	(01)/(10)	(01)/(10)	(00)/(11)	(01)/(10)	(01)/(10)	(01)/(10)	(00)/(11)

where "$\longrightarrow$" means that an error indication at the left causes an error indication at the right.

From Table 6.3, we can see that any single stuck-at fault is only indicated at its own error indicator.

6.1.3 The design of TR-TSC circuits with separate internal fault indication IF

TSC circuits with independent IF can be easily achieved using decoupling BBB DC_2. We use a design of TSC FA with separate IF to demonstrate the design technique of TSC circuits with separate IF.

An internal construction of a TR-TSC FA with separate IF is shown in Figure 6.6. Suppose that there is only one kind of error which can occur during normal operation. When the proposed FA receives error inputs, only EI indicates the faults. When there is any single internal stuck-at fault, IF produces an error indication.

Theorem 20 : *The proposed full adder is a TSC full adder.*

Proof: It is similar to the proof of **Theorem 17**. Here, it will not be repeated.

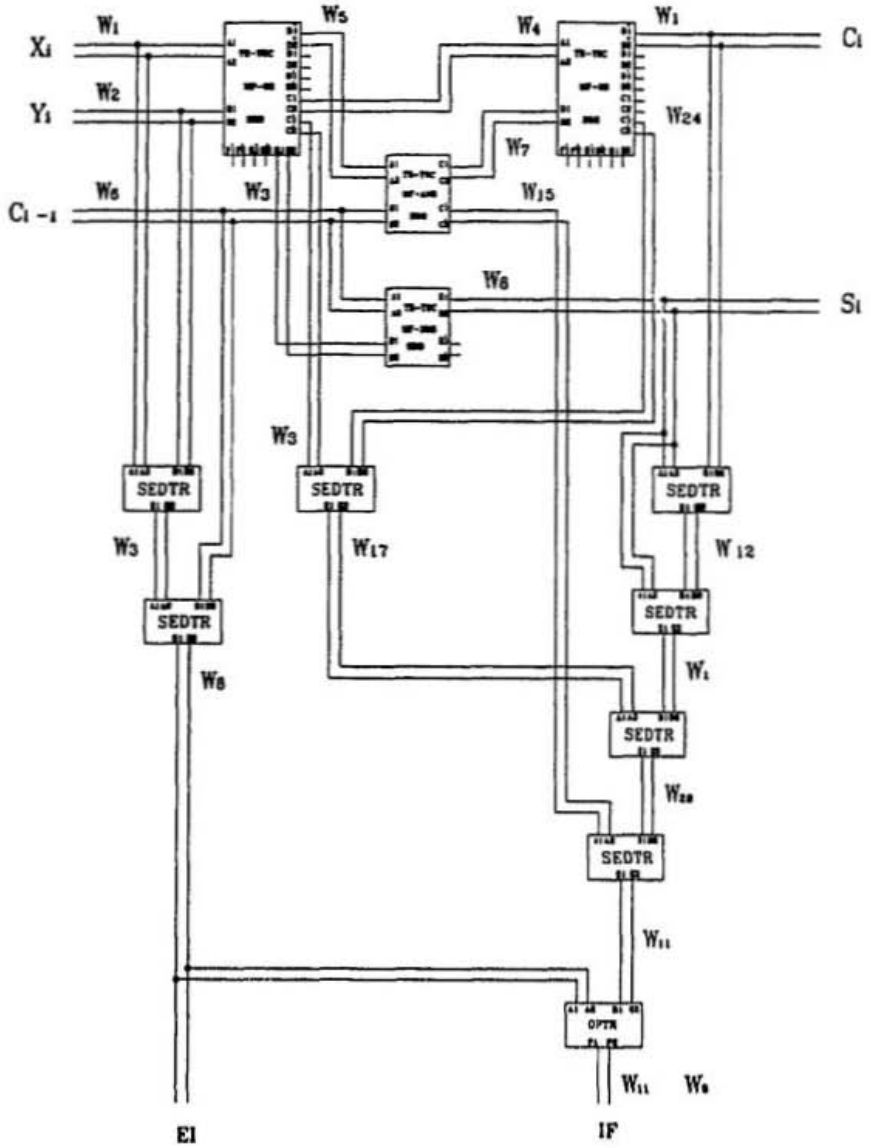

Figure 6.1: A structure of TR-TSC FA using TR-TSC MF BBBs.

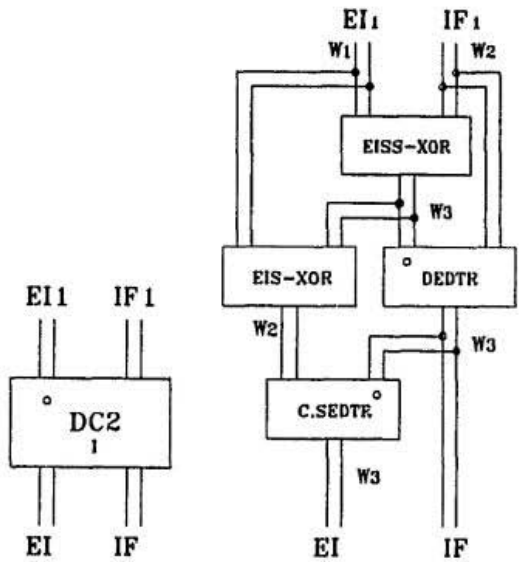

Figure 6.2: The internal construction of Type I DC_2.

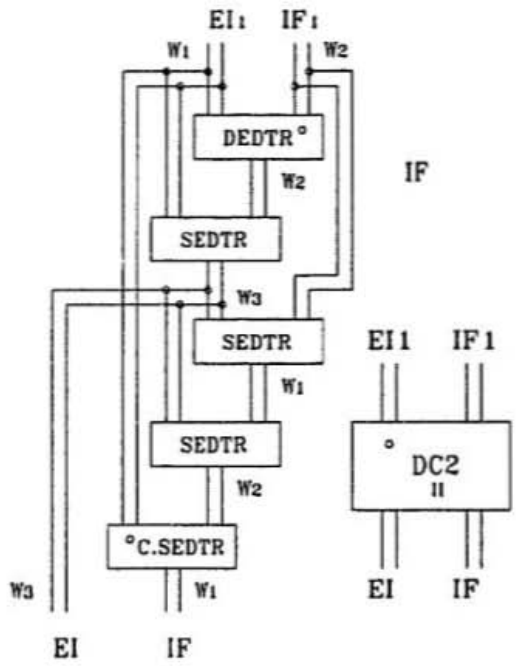

Figure 6.3: The internal construction of Type II DC_2.

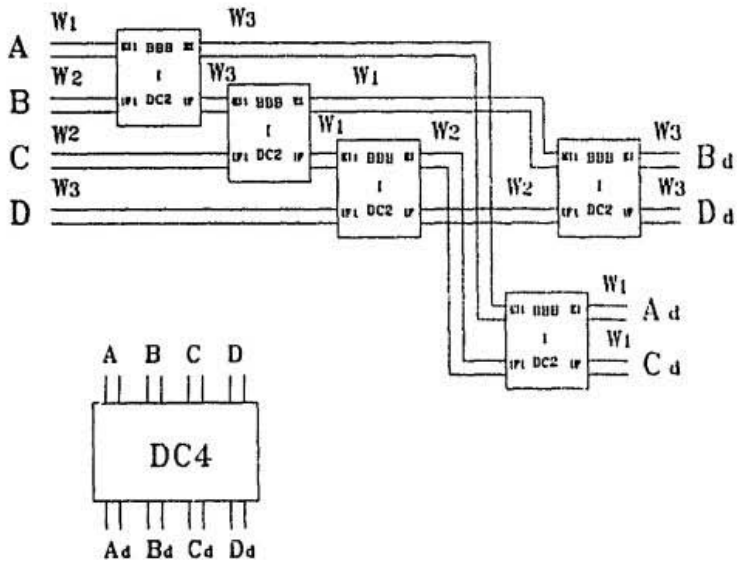

Figure 6.4: The internal construction of DC_4.

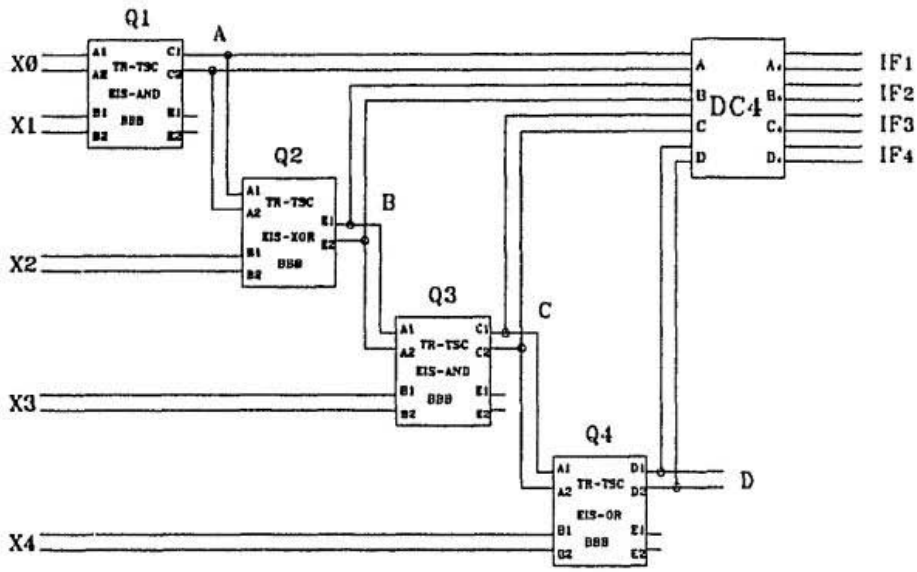

Figure 6.5: An example application of DC_4.

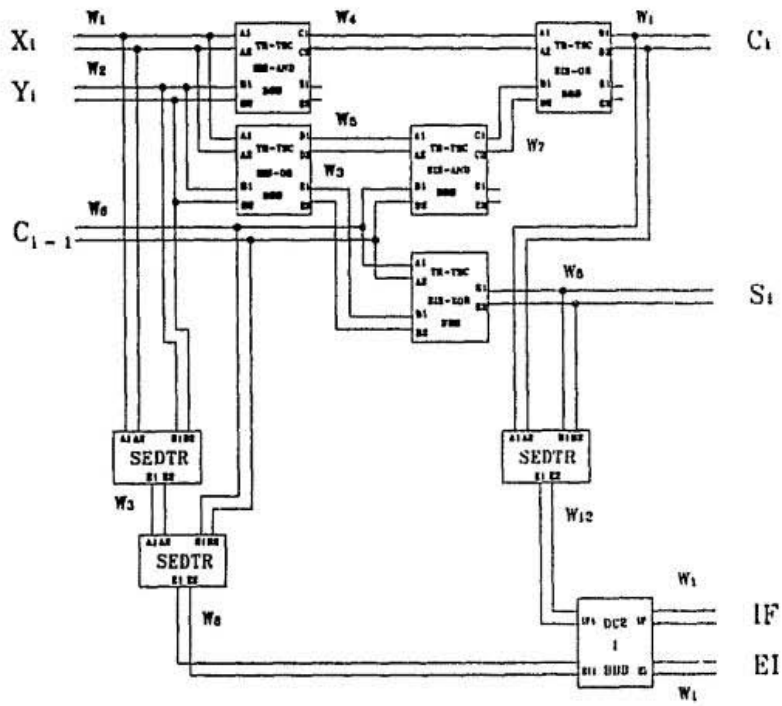

Figure 6.6: An internal construction of TR-TSC FA with separate IF.

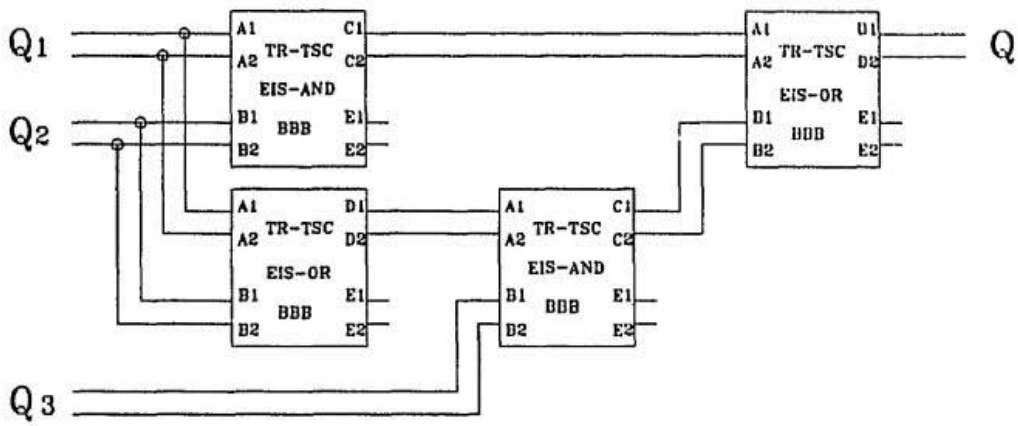

Figure 6.7: A two-rail voter (VT) with three inputs by SIM.

6.2 Image design method of TR-TSC functional circuits using BBBs

In many cases, SIM cannot achieve TR-TSC functional circuits. For example, a two-rail voter (VT) with three inputs using BBBs, shown in Figure 6.7, is constructed using SIM. VT implements the following function:

$$Q = Q_3(Q_1 + Q_2) + Q_1Q_2 \tag{6.5}$$

where Q_1, Q_2, Q_3 and Q are two-element variables.

Obviously, this VT possesses the *fault secure* property, for it either keeps the correct value or gives at least one error indication at its output Q when any error input or any single internal stuck-at fault occurs during normal operation. Since Q_1, Q_2, and Q_3 come from three identical TSC modular units and always give the same logic values (01) or (10), each BBB in the VT can never receive the complete minimal test set T_S during normal operation. Thus, it is not *self-testing*.

In order to overcome this problem, we propose an image design method (IDM) with a pair of complement translators.

Table 6.4: The truth table of CTHS and CTLS.

	CTHS				CTLS			
Strobe	Input		Output		Input		Output	
Sb	A_1	A_2	A_{H1}	A_{H2}	A_1	A_2	A_{L1}	A_{L2}
0	0	1	0	1	0	1	1	0
1	0	1	1	0	0	1	0	1
0	1	0	1	0	1	0	0	1
1	1	0	0	1	1	0	1	0
0	0	0	0	0	0	0	1	1
1	0	0	1	1	0	0	0	0
0	1	1	1	1	1	1	0	0
1	1	1	0	0	1	1	1	1

6.2.1 TSC complement translators – CTHS and CTLS

Two TSC complement translators are introduced here. One translator complements its inputs when its strobe Sb is at high level. We call it TSC complement translator with high strobe (CTHS). The other is just the opposite. It complements its inputs when strobe Sb is low. We call it TSC complement translator with low strobe (CTLS).

Their internal constructions are shown in Figure 6.8 and their truth table is given in Table 6.4.

Their TSC properties are demonstrated in Table 6.5 and Table 6.6, respectively.

Table 6.5: The verification of the TSC property for CTHS.

			stuck-at fault at gate-1				stuck-at fault at gate-2			
Strobe	input		stuck-at-0 at gate-1		stuck-at-1 at gate-1		stuck-at-0 at gate-2		stuck-at-1 at gate-2	
Sb	A_1	A_2	A_{H1}	A_{H2}	A_{H1}	A_{H2}	A_{H1}	A_{H2}	A_{H1}	A_{H2}
0	0	1	0	1	1	1	0	0	0	1
1	0	1	0	0	1	0	1	0	1	1
0	1	0	0	0	1	0	1	0	1	1
1	1	0	0	1	1	1	0	0	0	1
0	0	0	0	0	1	0	0	0	0	1
1	0	0	0	1	1	1	1	0	1	1
0	1	1	0	1	1	1	1	0	1	1
1	1	1	0	0	1	0	0	0	0	1

Table 6.6: The verification of the TSC property for CTLS.

			stuck at fault at gate 1				stuck-at fault at gate-2			
Strobe	input		stuck-at-0 at gate-1		stuck-at-1 at gate-1		stuck-at-0 at gate-2		stuck at-1 at gate-2	
Sb	A_1	A_2	A_{L1}	A_{L2}	A_{L1}	A_{L2}	A_{L1}	A_{L2}	A_{L1}	A_{L2}
0	0	1	0	0	1	0	1	0	1	1
1	0	1	0	1	1	1	0	0	0	1
0	1	0	0	1	1	1	0	0	0	1
1	1	0	0	0	1	0	1	0	1	1
0	0	0	0	1	1	1	1	0	1	1
1	0	0	0	0	1	0	0	0	0	1
0	1	1	0	0	1	0	0	0	0	1
1	1	1	0	1	1	1	1	0	1	1

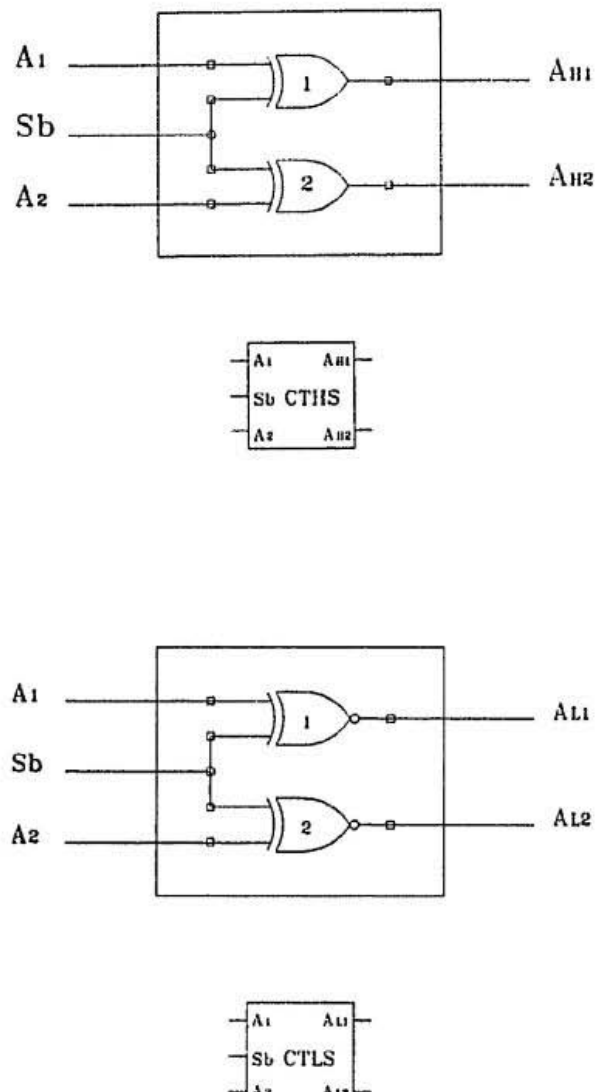

Figure 6.8: The proposed CTHS and CTLS.

6.2.2 Image design method (IDM)

A general design method called image design method (IDM) is introduced in this section. This method is a universal technique to design TR-TSC circuits using BBBs. In order to make each component receive the complete minimal testing set T_S, a pair of complement translators – CTHS and CTLS – is employed.

In order to achieve the TSC goal with IDM, the minimum requirement for a circuit which implements the given logic is that the inputs of the circuit do not keep only one logic level during normal operation.

A general structure of a TR-TSC circuit designed by IDM is shown in Figure 6.9. It is still composed of two parts – a TSC functional circuit and a TSC checker. Further, the TSC functional circuit has three different parts. They are a TR-TSC functional circuit with CTHSs (FC-L), a TR-TSC functional circuit with CTLSs (FC-H), and a TR-TSC multiplexer (MUX). Their functions during normal operation are described below.

1. FC-L: When Sb is low level, it implements given logic functions. When Sb is high level, it no longer gives correct functional values but proceeds with error diagnosis.

2. FC-H: Its function is just the opposite of FC-L. When Sb is low level, it proceeds with error diagnosis. When Sb is high level, it implements given logic functions.

3. MUX: It produces the outputs of the TSC circuit. While Sb is low level, it selects the outputs of FC-L as the outputs of the TSC circuit; while Sb is high level, it takes the values of FC-H as the outputs of the TSC circuit.

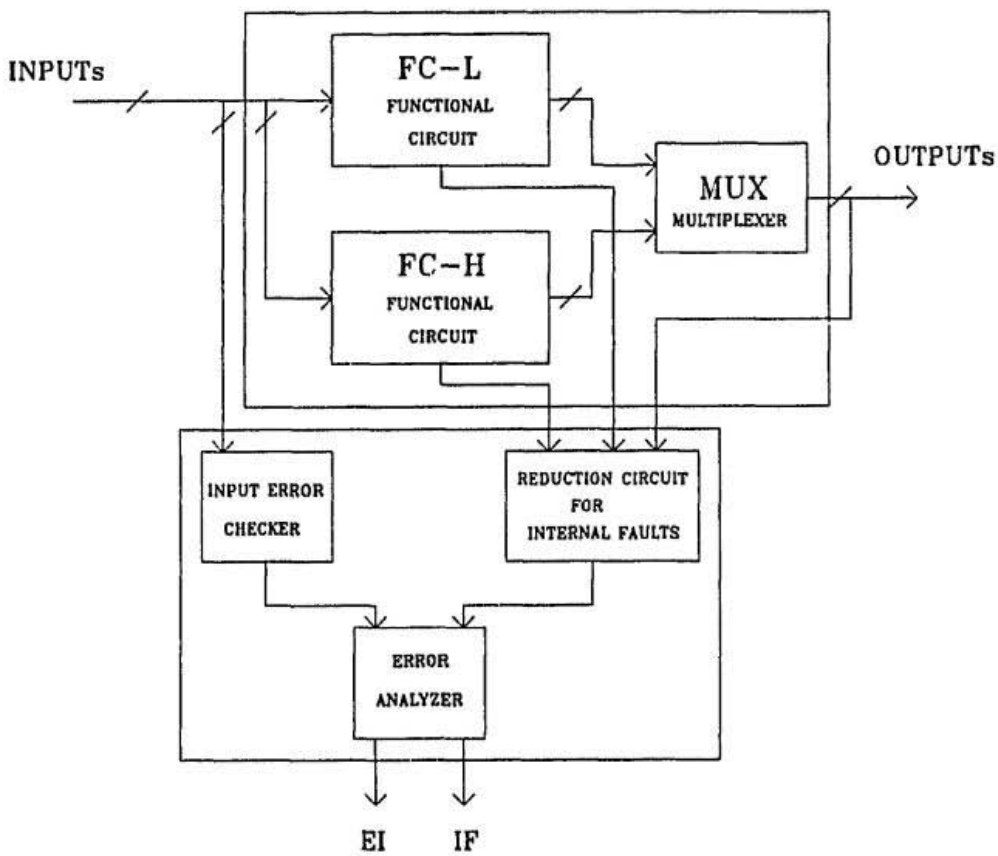

Figure 6.9: A general structure of TR-TSC circuits with IDM.

The TSC checker consists of a TSC input checker, a TSC reduction circuit for internal fault propagation, and TSC error analyzer. The input checker monitors error inputs. The reduction circuits provides error propagation paths for internal faults. The error analyzer distinguishes between any input error and any internal fault from a prescribed set of faults.

Since this structure has two symmetric functional circuits — FC-L and FC-H, we call the structure as image structure. Consequently, the design method employed in the design of this kind of TSC circuits is called Image Design Method (IDM).

The IDM is formalized as:

1. FC-L: Place CTHSs at one of the two inputs for all BBBs.
2. FC-H: Place CTLSs at one of the two inputs for all BBBs.
3. MUX: MUX is an EIS-XOR BBB. We locate a CTHS (CTLS) at its output and a CTHS (CTLS) at one of its inputs which is connected to FC-L (FC-H).
4. The TSC checker: Investigate each pair of monitored internal lines. When it produces only one value (01) or (10) during normal operation, place a complement translator at that place.

Theorem 21 : *The circuit constructed using IDM is a TSC circuit.*

Proof: The resulting circuit maps code inputs into code outputs and noncode inputs into noncode outputs during fault-free operation. Thus, it is *code-disjoint.* Since the circuit either keeps the correct value or gives at least one error indication at its error indication outputs EI and IF when any error input or any internal fault occurs during normal operation, it possesses the *fault secure* property. Because

each block is able to receive at least two testing vectors out of four testing vectors in T_S when Sb is at low level, and receive at least the other required testing vectors when Sb is at high level, the *self-testing* property is also guaranteed. □

Any combinational logic can be implemented by this design method. Let us consider an example of a TR-TSC voter which cannot be achieved by SIM. A group of simplified symbols of BBBs, CTHS and CTLS is given Figure 6.10. The proposed TR-TSC VT is shown in Figure 6.11.

Let Q_{ML} stands for the majority value determined by FC-L, and Q_{MH} for the majority value determined by FC-H, Q is the output of the circuit; then $Q = Q_{ML}$ and $Q_{MH} = 1$ when Sb is low level, and $Q = Q_{MH}$ and $Q_{ML} = 1$. Thus, Q always achieves the majority values.

Theorem 22 : *The proposed voter is a TSC voter.*

Proof: The proposed VT maps code inputs into code outputs and noncode inputs into noncode outputs during fault-free operation. Thus, it is *code-disjoint.* Since the circuit either keeps the correct value or gives at least one error indication at its error indication outputs EI and IF when any error input or any internal fault occurs during normal operation, it possesses the *fault secure* property. Because each block is able to receive at least two testing vectors out of four testing vectors in T_S when Sb is low level, and receive at least the other required testing vectors when Sb is high level, the *self-testing* property is also guaranteed. □

Its internal fault indication IF is independent of error input indication EI.

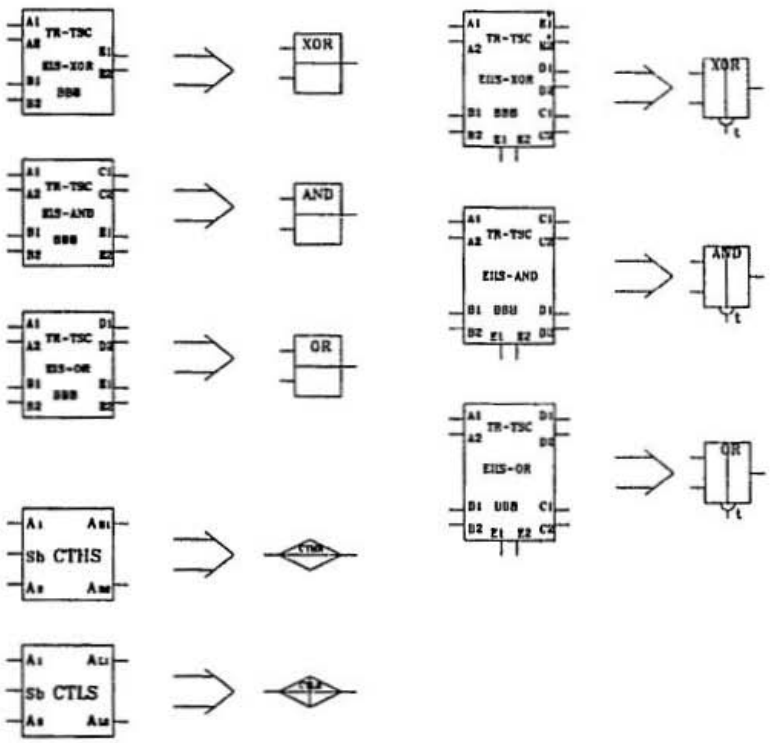

Figure 6.10: The simplified symbols for BBBs, CTHS and CTLS.

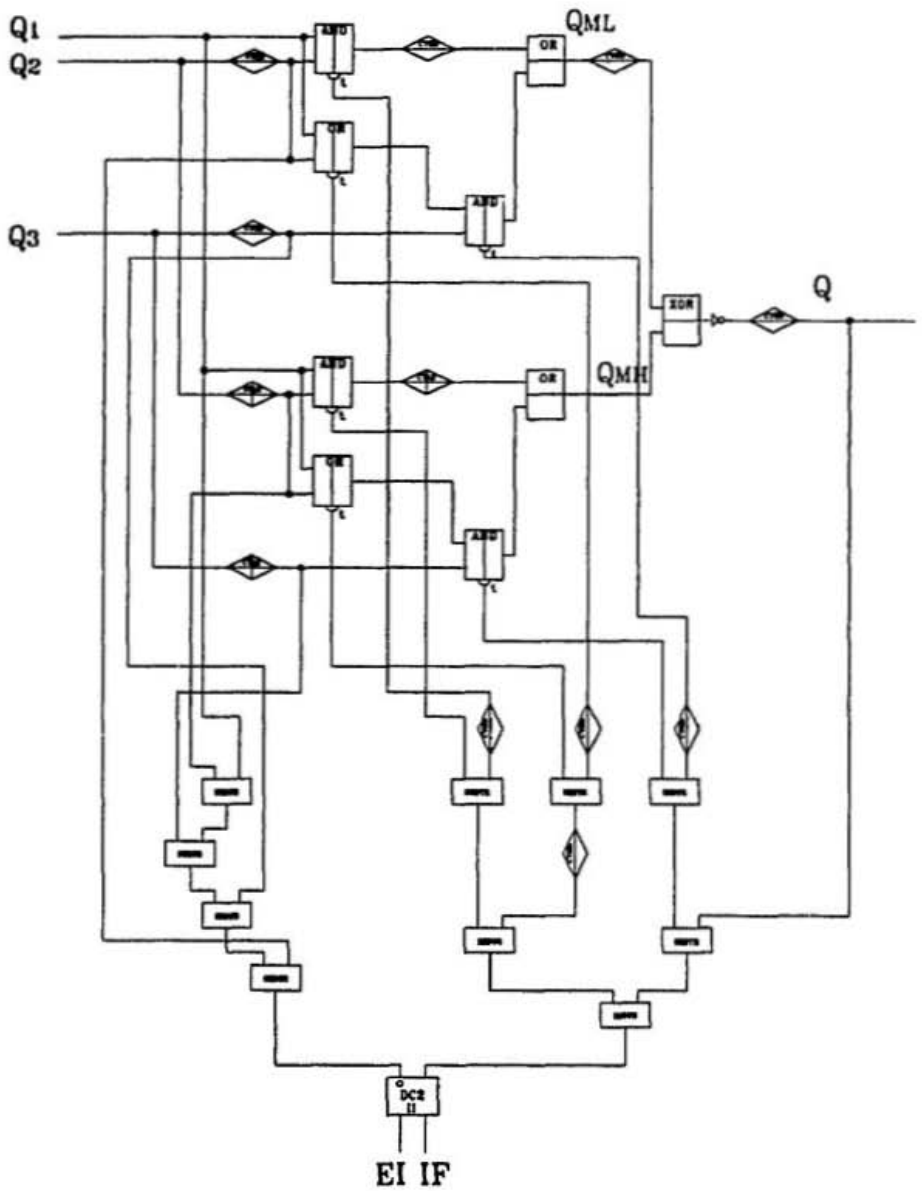

Figure 6.11: A structure of the proposed TR-TSC voter.

6.3 A new generation of TSC circuits — TSC error-confining (ECF) circuits

Ordinary TSC circuits have the capability of concurrent error detection but do not have the capability of confining internal faults in several particular areas. This leads to a low localizability and availability, and a high cost for maintenance.

A new generation of TSC circuits called TSC error-confining (ECF) circuits is introduced here

6.3.1 The definition of TSC error-confining circuits

We define an error-confining circuit as:

Definition 13 : *A TSC circuit is called a TSC error-confining (ECF) circuit if and only it is capable of confining internal faults from a prescribed set of faults in separate areas.*

According to the above definition, ECF circuits has many independent areas in teams of error detection. When any internal fault occurs, it will be isolated in a particular area. Each area has its own error indicator IF_i. If any internal fault occurs in i^{th} area, it is only indicated by IF_i. Each IF_i is independent of each other and also independent of error-input indication EI.

6.3.2 The design of TSC ECF circuits

In order to form independent error-isolation areas, EIIS BBBs are properly used to build *isolation boundaries.*

We give an example to demonstrate the configuration of TSC ECF circuits.

Example: An arbitrarily given TR-TSC circuit consisting of BBBs and CTHSs and CTLS is given in Figure 6.12. Suppose that the inputs of each BBB in this circuit are connected to some previous outputs or inputs and its output is provided to subsequent BBBs or as one of outputs of the circuit.

Now, all the BBBs composed of column 1 and 6, as well as row 8 are used with corresponding EIIS BBBs so that the circuit still keeps the logic functions during fault-free operation. The rest of BBBs are EIS BBBs. These placements using EIIS BBBs, shown in Figure 6.13, form several isolation boundaries. These boundaries partition the circuit into four independent parts (Part 1, Part 2, Part3, and Part 4) in a sense of error detection. The proposed partition is illustrated in Figure 6.14.

These four independent parts have their own independent internal fault indications — IF_1, IF_2, IF_3, and IF_4. If any internal fault occurs, it will be only indicated at its corresponding IF_i. For instance, if any single stuck-at fault or unidirectional faults occur in Part 2 and there is no fault in the rest of the circuit, the faults in Part 2 only indicated at IF_2 by (00) or (11). If all four checkers produce (00) or (11), this means that all four independent parts have internal faults. The resulting ECF circuit which has four independent error-isolation areas is shown in Figure 6.15.

Here, internal fault indication IF is independent of error input indication EI. Thus, the ECF circuit possesses a high capability of fault tolerance, a high localibility, and a high maintainability. Of course, all these advantages are realized by providing additional hardware. The amount of hardware required here is more than what is required in the other design procedured, but this method results in circuits that enable error location.

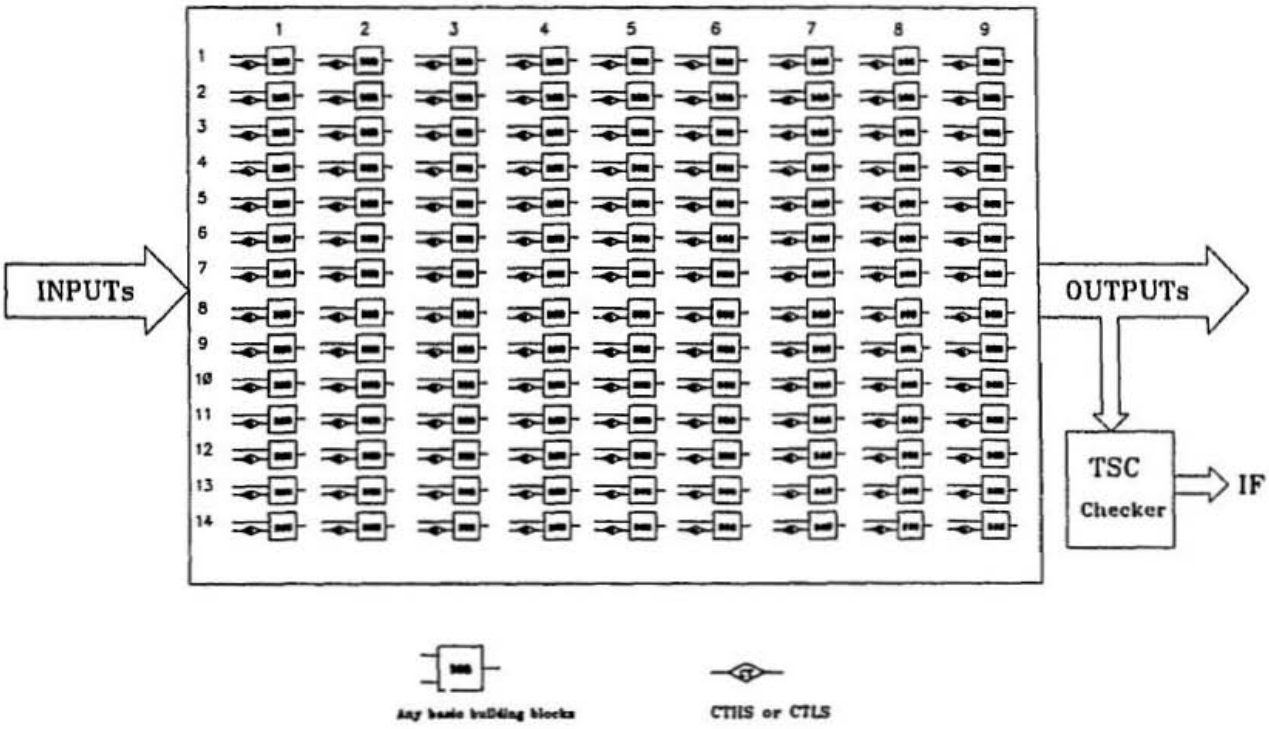

Figure 6.12: An arbitrary TR-TSC circuit using BBBs.

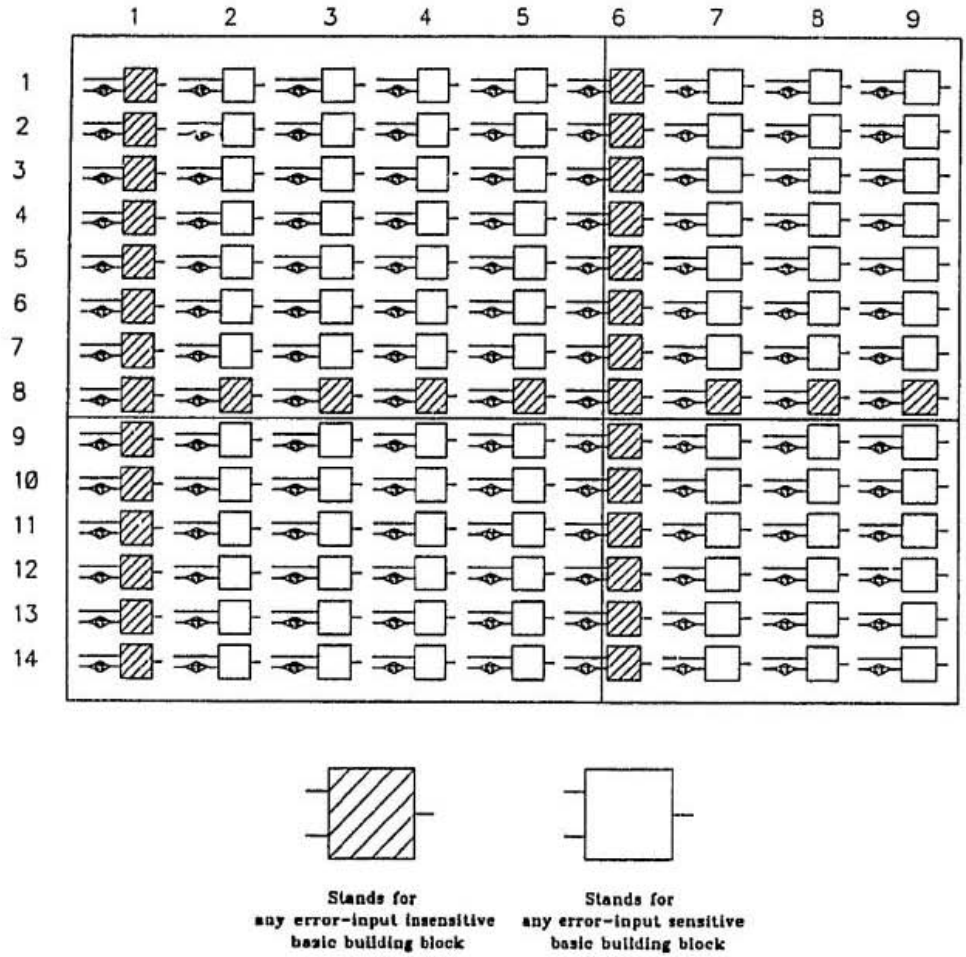

Figure 6.13: The proposed placements of EIIS BBBs and EIS BBBs for the given circuit.

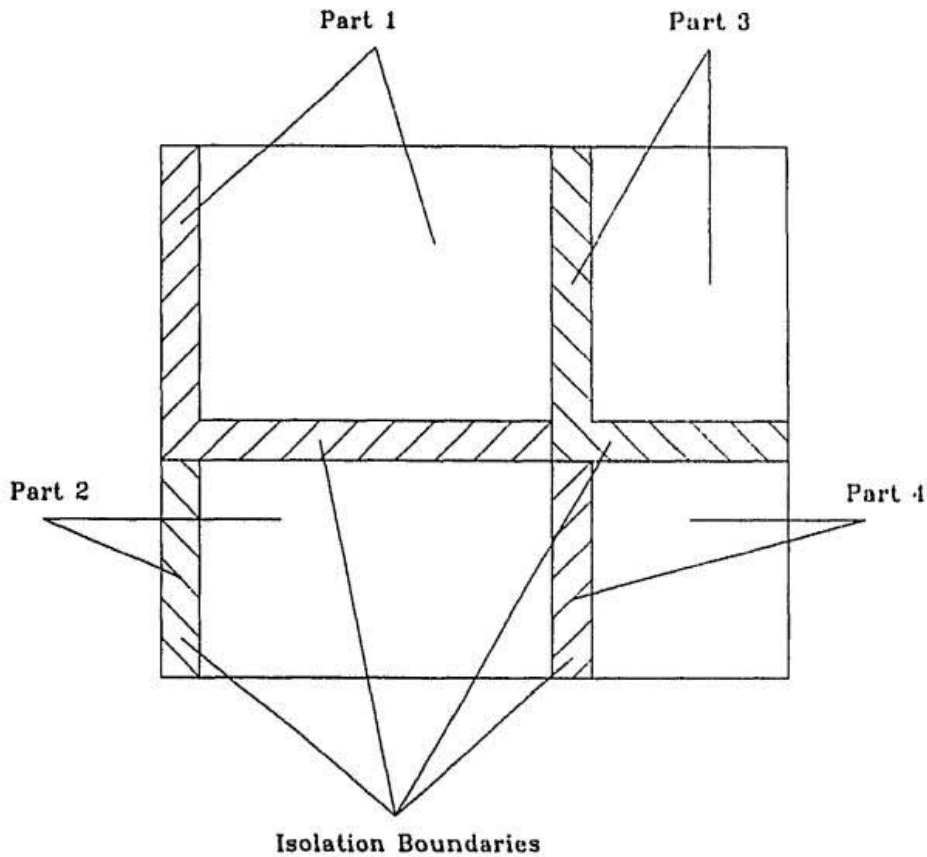

Figure 6.14: A proposed partition for the given circuit with four separate parts.

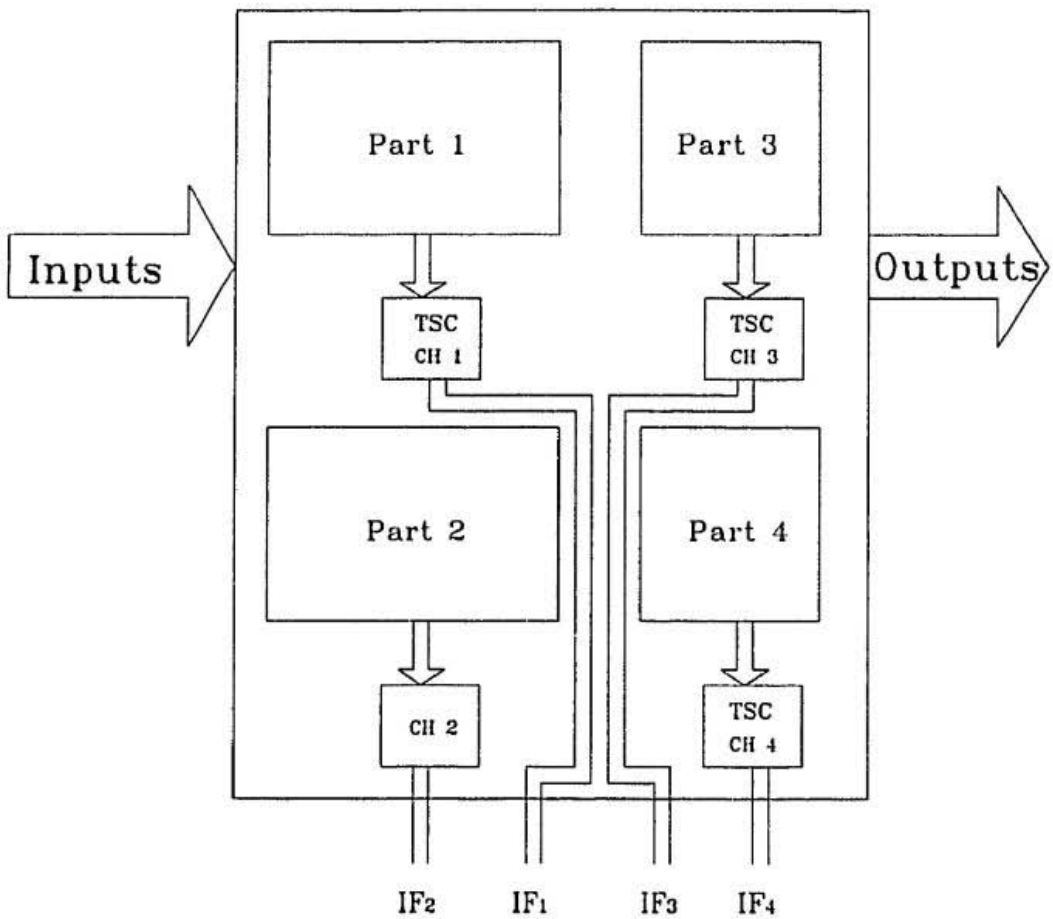

Figure 6.15: The resultant TSC ECF circuit with four independent IF.

Table 6.7: The truth table of TSC DIDC BBB.

EI_1	EI_1	IF_1	EI_1^*	EI_2^*	IF
(01)/(10)	(01)/(10)	(01)/(10)	(01)/(10)	(01)/(10)	(01)/(10)
(00)/(11)	(01)/(10)	(00)/(11)	(00)/(11)	(01)/(10)	(01)/(10)
(01)/(10)	(00)/(11)	(00)/(11)	(01)/(10)	(00)/(11)	(01)/(10)
(00)/(11)	(00)/(11)	(01)/(10)	(00)/(11)	(00)/(11)	(01)/(10)

6.3.3 A note of isolation boundary

Isolation boundaries are composed of EIIS BBBs. Any single stuck-at fault in an EIIS BBB is reflected at its two outputs. One is its main output (i.e. EIIS-AND BBB is output C), and the other is the semi-sensitive output E (test output). Since this semi-sensitive output E reflects one of its two inputs in a sense of error detection, if we simply use EIIS BBBs to construct isolation boundaries, a part of internal faults would pass the isolation boundaries and make its IF_i produce an error indication when any fault occurs in its previous neighboring area A_{i-1}.

In order to overcome this undesirable case, we develop another special TR-TSC BBB called double-input decoupling (DIDC) BBB.

A proposed structure of DIDC BBB is shown in Figure 6.16. It consists of two DC_2. Its truth table is given in Table 6.7.

Theorem 23 : *The proposed DIDC BBB is a TSC BBB.*

Proof: The proposed DIDC BBB possesses the *fault secure* property, for it either keeps the correct value or gives at least one error indication at its output IF when any internal fault occurs during normal operation. The *self-testing* property is also preserved. This has been verified by CDS illustrating in Figure 6.16. □

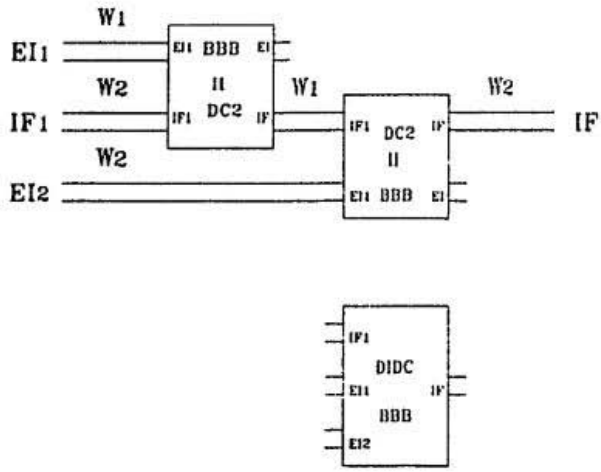

Figure 6.16: The proposed TSC DIDC basic building block.

To construct an isolation boundary, we should consider two cases. One is that each EIIS BBB in the isolation boundary is independent of each other. The other is that some BBBs in the isolation boundary are not independent of each other.

For the independent case, we give the design method which is shown in Figure 6.17. The isolation boundary is composed of three EIIS BBBs and three DIDC BBBs and two SEDTR blocks. Any input error is not reflected at IF_b. IF_b only indicates the faults which occurs in three DIDC BBBs, two SEDTR blocks, and a part of three EIIS. For the dependent case, we give the design method which is shown in Figure 6.18. Similarly, any input error is not reflected at IF_b. IF_b only indicates the faults which occurs in the isolation boundary itself.

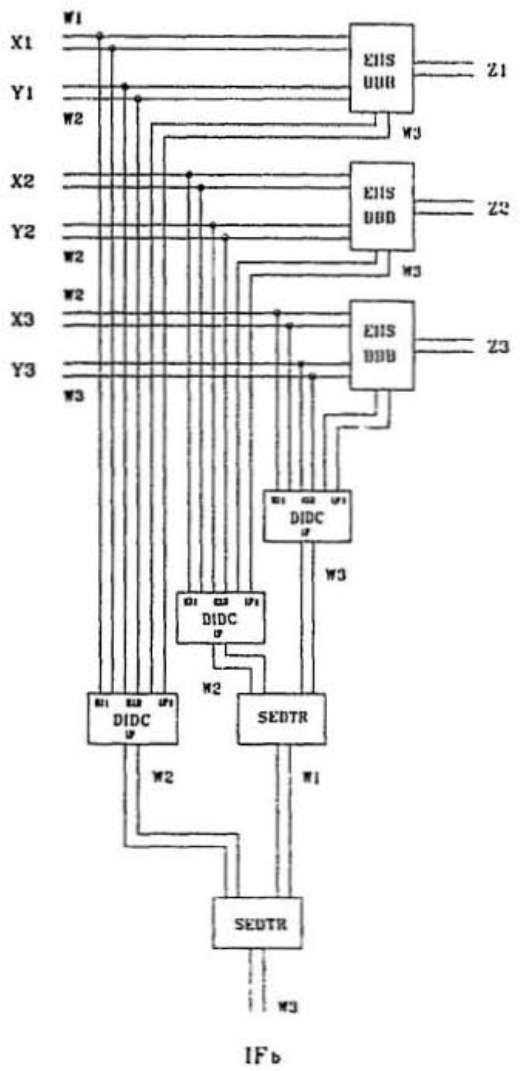

Figure 6.17: A structure of isolation boundary with independent EIIS BBBs.

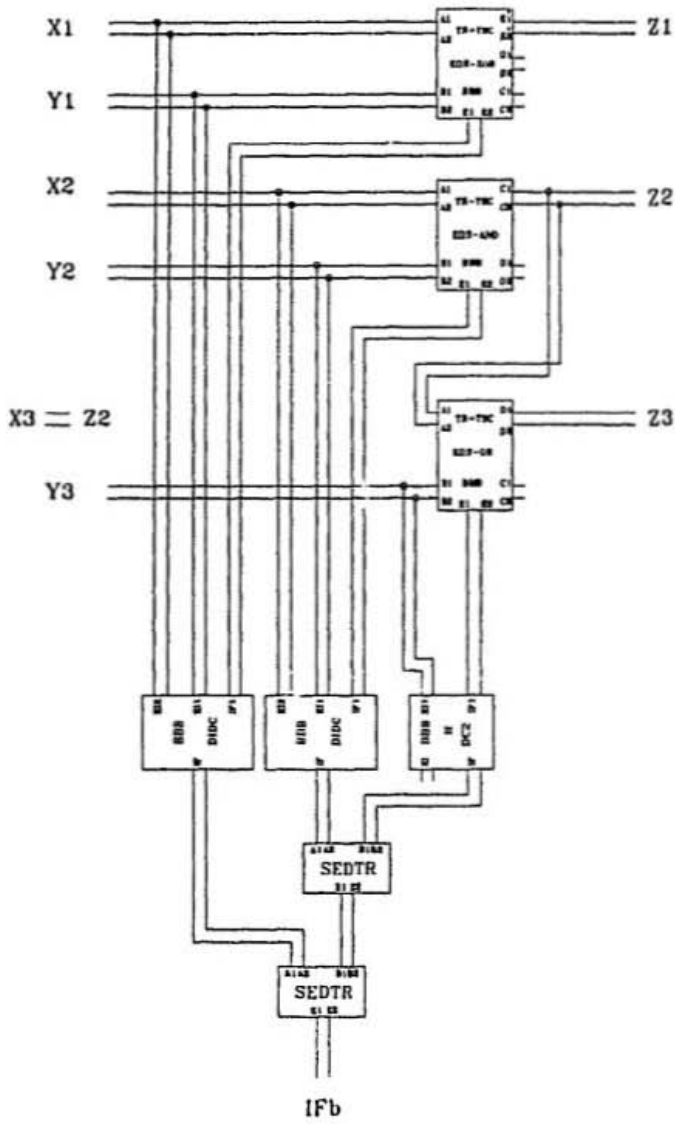

Figure 6.18: A structure of isolation boundary with dependent EIIS BBBs.

6.4 A brief discussion on sequential basic building blocks

Designing a set of sequential BBBs appears very difficult. There has been no reported work on the sequential BBBs which can be used in designing TSC sequential circuits. Conventional schemes of fault tolerance for sequential circuits can be classified into two main methods — fault-masking scheme and state-coding scheme. Although these two schemes can be implemented with TSC technique, their complicated structure makes it very difficult to locate the faulty units and replace them. Once internal faults occur, the whole circuit has to be disregarded. Moreover, common sequential techniques cannot be directly applied to the design of TSC sequential circuits. This leads to a high cost of maintenance and a low availability. Therefore, it is also desirable to develop a set of universal sequential BBBs.

6.4.1 A TSC D flip-flop

As one of the most important sequential components, D flip-flop is widely used in designing sequential circuits which implement given sequential logic functions.

Unlike the ordinary D flip-flop which can be designed using common logic gates, the TSC D flip-flop cannot be achieved using TR-TSC BBBs. If we develop a TR-TSC D flip-flop using BBBs and using the similar technique as for ordinary D flip-flop, the resulting D flip-flop will never work. The reason is that the resulting D flip-flop cannot get out of its initial state (00). This means the flip-flop cannot start automatically.

A TSC D flip-flop based on the TMR technique [16] is achieved. Figure 6.19 is

its internal construction. IF depends only on masked faults in TMR system and all the faults in the error detecting circuit. S indicates unmasked faults in the TMR system as well as a small and constant number of faults in the error detecting circuit. IF is a warning signal and S is a stop signal.

Theoretically, the proposed TSC D flip-flop can be used as a sequential BBB to design TSC sequential circuits. Considering its hardware cost and its gate levels, it is not an economic and efficient BBB. To develop a practical TSC D flip-flop is a pending and challenging problem.

6.5 Concluding remarks

In this chapter, we have proposed two design methods — SIM and IDM — to build TR-TSC functional circuits using BBBs. A special BBB — DC_2 — has been developed. DC_2 is very useful for designing TSC circuits with separate IF. A decoupling technique has also been studied. We have introduced a new generation of TSC circuits — TSC ECF circuits. TSC ECF circuits have the capability of confining internal faults in separate areas. The DIDC BBB which has been presented in this chapter is a key component for building isolation boundaries. Finally, we have presented a brief discussion on sequential TSC BBB, and developed a TSC D flip-flop.

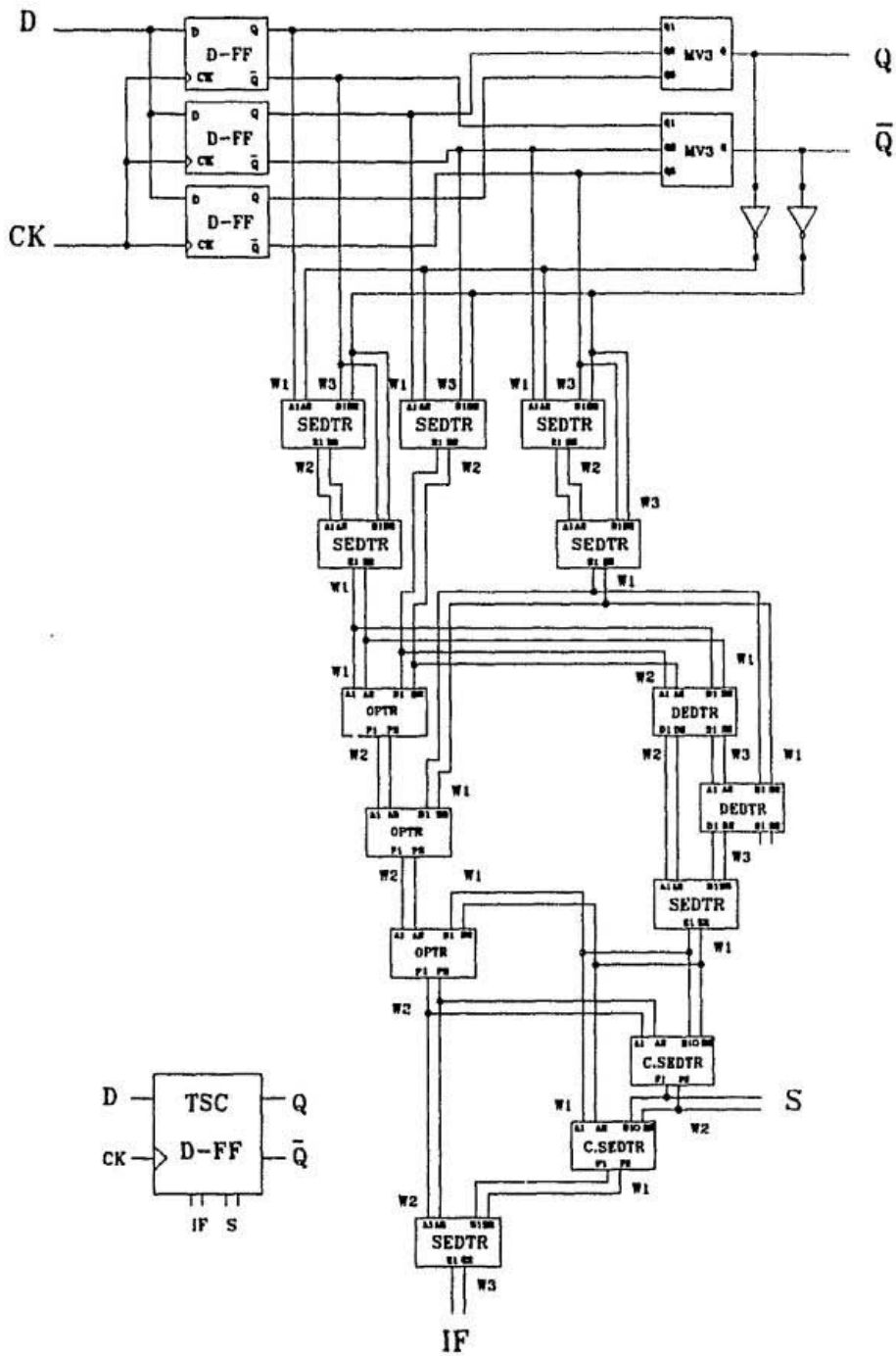

Figure 6.19: The proposed TSC D flip-flop.

Chapter 7

An efficient combinational TSC checker for 1-out-of-3 code

An indirect design technique for a combinational totally self-checking (TSC) 1-out-of-3 code checker is presented. In comparison to the existing indirect techniques to design combinational TSC 1-out-of-3 code checkers, the proposed one uses less hardware, has fewer gate levels, and possesses a higher test capability.

7.1 Design motivation

There have been many results on the problem of designing TSC checkers for 1-out-of-n codes, a special subclass of m-out-of-n code, with $n > 3$, but only a few of them relate to the $n = 3$ case [10], [17], [21], [25], [30]. Reddy [29] conjectured that no TSC checker (direct design) exists for the 1-out-of-3 code, especially if the checker is constructed from only AND and OR gates. So far, only two direct design techniques of the TSC 1-out-of-3 code checkers have been published, but they are both combinational NMOS implementations [21], [30]. Combinational gate-level implementations of the TSC 1-out-of-3 code checkers described in [17], [25] belong to indirect design techniques.

Based on the fact that 1-out-of-3 code has three codewords and these three codewords are not enough to constitute a complete test set for an independent combinational 1-out-of-3 code checker with respect to a set of all single stuck-at faults [29], the design techniques in [17] & [25] make use of at least one known TSC checker in a TSC system as their auxiliary conditions. The design methods presented by Golan in [17] need any available m-out-of-n code in a TSC system combined with 1-out-of-3 code to obtain a reduced $(m + 1)$-out-of-$(n + 3)$ code for which a TSC checker can be designed. The design methods proposed by Paschalis *et al* in [25] also follow the basic idea that combines the 1-out-of-3 code with other codes although the codes are no longer restricted to fixed-weight codes. However, a translator T first translates the 1-out-of-3 code into an incomplete two-variable two-rail code. Then, the incomplete two-variable two-rail code is properly combined with the 1-out-of-2 code outputs of one or more other TSC checkers which are available in a TSC system.

In this chapter, we present a combinational TSC 1-out-of-3 code checker with respect to all single stuck-at faults. The basic idea to translate the 1-out-of-3 code into an incomplete two-variable two-rail code [25] is also followed. We make use of the complement translator (CTHS) to provide the other necessary test input (01, 01) for the reduction circuit which is a TR-TSC EIS-XOR BBB. Thus, the TR-TSC EIS-XOR BBB can receive its complete test set which consists of four test inputs – (01, 01), (01, 10), (10, 01), and (10,10) – during normal operations.

The terms and definitions employed here are based on those used in [25].

7.2 A proposed design method

With the help of any TSC checker available in a TSC system, the TSC 1-out-of-3 code checkers have been achieved in [17], [25]. But any input error or internal fault from a prescribed set in the known TSC checker will lead to an error indication at its 1-out-of-2 code output. In that case, the resulting TSC 1-out-of-3 code checker cannot receive all required test inputs. Thus, the test capability of TSC 1-out-of-3 code checkers is affected by the known TSC checker, especially when the known TSC checker is very large.

Here, we present a simpler design which enhances the test capability and uses less hardware and has fewer gate levels. The 1-out-of-3 code is also initially translated into an incomplete two-variable two-rail code by the TSC translator T [25]. Then, a TSC complement translator with high strobe (CTHS) is placed at the output (a1, b1) of T. The output (Z1, Z2) of **CTHS** and the output (a2, b2) of T together form the inputs to the TR-TSC EIS-XOR BBB. Thus, the TR-TSC EIS-XOR BBB receives its test inputs (10, 01), (01, 10), and (10, 10) when Sb is low, and receives (01, 01), (10, 10), and (01, 10) when Sb is high. Consequently, the TR-TSC EIS-XOR BBB will receive the complete test set during normal operations.

The auxiliary input Sb can be any independent line in a system, which has a logic 1 level and a logic 0 level during the normal operation. For example, we can choose system clock as the auxiliary line Sb.

The gate-level implementations of the translator T is shown in Figure 7.1. Its truth table is given in Table 7.1.

The logic gate implementation of the proposed TSC checker for 1-out-of-3 code is shown in Figure 7.2 and its truth table is given in Table 7.2.

Table 7.1: The truth table of the translator T.

Input			Output			
X1	X2	X3	a1	b1	a2	b2
0	0	0	0	0	0	0
0	0	1	1	0	0	1
0	1	0	0	1	1	0
1	0	0	1	0	1	0
0	1	1	1	1	1	1
1	0	1	1	0	1	1
1	1	0	1	1	1	0
1	1	1	1	1	1	1

Theorem 24 : *The described checker above is a TSC checker for 1-out-of-3 code.*

Proof: The proposed checker maps code inputs into code outputs and noncode inputs into noncode outputs during fault-free operation. Thus, it is *code-disjoint.* Since the checker either keeps the correct value or gives an error indication when any single internal stuck-at fault occurs during normal operation, it possesses the *fault secure* property. It is obvious that the *self-testing* property is also reserved, for each component– T, CTHS, and EIS-XOR BBB – receives its own all test inputs. Consequently, the checker is a TSC checker. □

7.3 Comparisons

The proposed TSC 1-out-of-3 code checker uses only 10 logic gates and four gate-level delay. Although an auxiliary line Sb is introduced, the probability of stuck-at faults at a single line is much less than in a complicated circuit. Therefore, the proposed TSC checker has a higher test capability than the TSC checkers described in [17], [25]. If the known TSC checker is considered, the presented TSC checkers in [25] has, in the most favorable case, at least 20 logic gates and six gate-

Table 7.2: The truth table of the proposed TSC checker for 1-out-of-3 code.

Sb	X1	X2	X3	a1	b1	Z1	Z2	a2	b2	f	g
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	0	0	1	0	1
0	0	1	0	0	1	0	1	1	0	0	1
0	1	0	0	1	0	1	0	1	0	1	0
0	0	1	1	1	1	1	1	1	1	1	1
0	1	0	1	1	0	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	0	0	0	0
1	0	0	1	1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	0	1	0	1	0
1	1	0	0	1	0	0	1	1	0	0	1
1	0	1	1	1	1	0	0	1	1	0	0
1	1	0	1	1	0	0	1	1	1	1	1
1	1	1	0	1	1	0	0	1	0	0	0
1	1	1	1	1	1	0	0	1	1	0	0

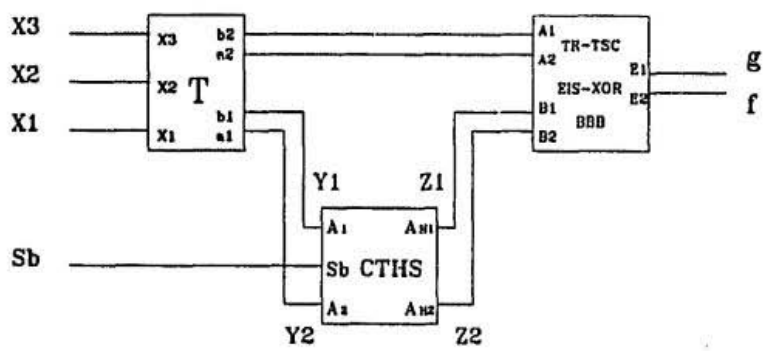

Figure 7.2: The logic gate implementation of the proposed TSC checker for 1-out-of-3 code.

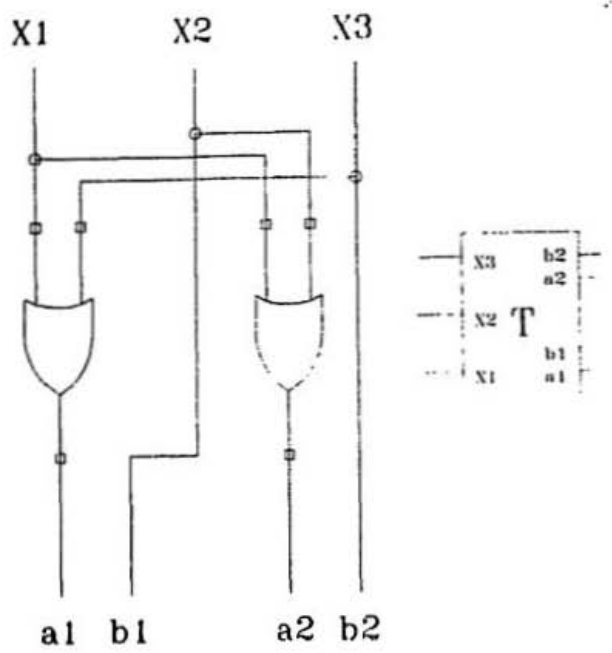

Figure 7.1: The gate-level implementations of the translator T.

level delay. In addition, it has at least 7 input lines compared to 4 input lines of the TSC checker proposed here. It is evident that the proposed TSC 1-out-of-3 code checker is superior to the existing TSC 1-out-of-3 code checkers in [17], [25].

7.4 Concluding remarks

The design method for the TSC 1-out-of-3 code checker proposed in this chapter is the simplest indirect design method for the combinational implementation at logic gate level. It employs less hardware and fewer gate-level delay. As it does not depend on any known TSC checker in a TSC system, it has a higher test capability.

Chapter 8

Summary and Suggestions for Future Research

8.1 Summary

In this thesis, we have presented three types of TR-TSC BBBs. These BBBs, like ordinary logic gates in common digital circuits, can be easily applied to the design of TSC circuits. Common logic design methods, e.g., K-map simplification technique, can be directly used in the design of TSC circuits. The formalizations of strong morphic Boolean algebra B_{SM} and the new classification of checkers lay down the theoretical foundation for the development of TR-TSC BBBs and the design of TR-TSC circuits using the TR-TSC BBBs.

According to B_{SM}, the logic representations of two-rail error indication variable are 01 for logic 0, 10 for logic 1, and 00 or 11 for error indication. Thus, TR-TSC circuits composed of BBBs not only can implement given logic functions but also have the capability of concurrent error detection.

Three types of TR-TSC BBBs — EIS BBBs, EISS BBBs, and EIIS BBBs have been described. Each type of BBBs consists of three basic function blocks

– AND-BBB, OR-BBB, and XOR-BBB. These universal BBBs are very useful for constructing TR-TSC functional circuits.

Two design methods — SIM and IDM — have been proposed. SIM employs less hardware but is suitable for the case where the *self-testing* property can be easily achieved or verified. The verification of self-testing property of circuits which are designed by SIM is difficult. On the other hand, IDM can deal with any case. It is a general design method but uses more hardware. The verification of self-testing property of circuits with IDM is quite simple.

A very useful BBB — a decoupling block (DC_2) has been proposed and two types of TR-TSC DC_2's have been developed. DC_2 is particularly useful for designing TSC circuits with separate EI and IF. A new class of TR-TSC circuits called TSC decoupling circuits has been studied. These decoupling circuits are mainly used to distinguish relevant error indication variables, and to locate fault sources.

A new generation of TSC circuits called error-confining (ECF) circuits has been introduced. In a ECF circuit, internal faults are confined in separate areas and indicated by independent internal fault indicators. These greatly improve maintainability, availability, and reliability. A special technique for designing isolation boundaries has been presented. This has been achieved using EIIS BBBs and a double-input decoupling BBB.

A structure of a TSC D flip-flop has been proposed. Theoretically, the proposed TSC D flip-flop can be used as a universal sequential BBB, combined with combinational BBB, to design any TSC sequential circuit. However, the proposed D flip-flop costs much hardware and has a low speed.

By means of the principle and concept of two-element morphic space theory,

an efficient TSC combinational checker for 1-out-of-3 code has been successfully achieved. The proposed one uses less hardware, has fewer gate levels, and possesses a higher test capability.

Finally, we can say that the proposed design methods for TR-TSC circuits using BBBs and existing operator blocks are simple, convenient, flexible, and systematic techniques. In addition, the proposed two-rail TSC circuits can detect not only single stuck-at faults but also unidirectional faults. Thus, they have a wider application potential. Some of the design techniques proposed here involve a high degree of redundancy, but we believe that the increased complexity of hardware will be offset by the benefits accrued in terms of fault tolerance.

8.2 Suggestions for Future Research

Future research could be carried out in the following aspects:

- Investigate characteristics of various TR-TSC circuits which are composed of different BBBs. Since the placements of different BBBs in TR-TSC circuits cause different flow of error indication propagation, error diagnosis depends mainly on the relevance of error indication variables. In general, the relevance of error indication variables in a TR-TSC circuit can be classified as relevant, irrelevant, semi-relevant, conditionally relevant. Systematically study the theory of relevance of error indication variables for TR-TSC circuits. Develop efficient circuit structure to enhance the capability of fault diagnosis and improve maintainability.

- Study efficient and practical schemes of error correction for the TR-TSC circuits which are composed of the proposed TR-TSC BBBs. Make new TR-

TSC circuits have the capability of correcting undesirable errors, as well as the capability of confining, locating their internal faults.

- Develop cost-effective TSC sequential components. Study theory and design methods of TR-TSC sequential circuits.
- Improve the proposed BBBs and achieve new BBBs which use less hardware and has fewer gate levels.
- Develop special BBBs to enhance reliability, availability, and maintainability.

References

[1] W.C. Carter and P.R. Schneider, *Design of dynamically checked computers*, in Proc. IFIP Congr. 68, vol. 2, Edinburgh, Scotland, 1968, pp [878 - 883]

[2] D.A. Anderson and G. Metze, *Design of totally self-checking check circuits for m-out-of-n codes*, IEEE Trans. Comput., vol. C-22, March 1973, pp [263 - 269]

[3] J. Wakerly, *Error detecting codes, self-checking circuits and applications*, 1978 Elsevier North-Holland

[4] P.K. Lala, *Fault tolerant & fault testable hardware design*, 1985 Prentice-Hall

[5] D.K. Pradhan, *Fault-tolerant computing – theory and techniques*, vol. I & vol. II, 1986 Prentice-Hall

[6] T.R.N. Rao and E. Fujiwara, *Error-control coding for computer systems*, 1989 Prentice-Hall

[7] D.A. Anderson, *Design of self-checking digital networks using coding techniques*, Urbana, CSL/Univ. of Illinois, Rep. 527, September 1971

[8] M.J. Ashjaee and S.M. Reddy, *On totally self-checking checkers for separate codes*, IEEE Trans. Comput., vol. c-26, No. 8, August 1977, pp [737 - 744]

[9] W.C. Carter, A.B. Wadia, and J.C. Jessep, *Implementation of checkable acyclic automata by morphic Boolean functions*, in Proc. Symp. Comput. Automata, Polytechnic Institute of Brooklyn, April 1971, pp [465 - 482]

[10] R. David, *Totally self-checking 1-out-of-3 code checker*, IEEE Trans. Comput., vol. C-27, June 1978, pp [570 - 572]

[11] N. Gaitanis, *Totally self-checking checkers for low-cost arithmetic codes*, IEEE Trans. Comput., vol. c-34, No. 7, July 1985, pp [596 - 601]

[12] N. Gaitanis, *The design of N-modular redundancy systems*, Proceedings 2nd International Conference on supercomputing, March 1987, pp [238 - 244]

[13] N. Gaitanis, *The design of TSC error C/D circuits for SEC/DED codes*, IEEE Trans. Comput., vol. c-37, No. 3, March 1988, pp [258 - 265]

[14] N. Gaitanis, *TSC-error C/D circuits for SEC/DED product codes*, IEE Proceedings, vol. 135pt. E, No. 5, September 1988, pp [253 - 258]

[15] N. Gaitanis, *Totally self-checking checkers with separate internal fault indication*, IEEE Trans. Comput., vol. 37, No. 10, October 1988, pp [1206-1213]

[16] N. Gaitanis, *The design of totally self-checking TMR fault-tolerant systems*, IEEE Trans. Comput., vol. c-37, No. 11, November 1988, pp [1450 - 1454]

[17] P. Golan, *Design of totally self-checking checker for 1-out-of-3 code*, IEEE Trans. Comput., vol. C-33, March 1984, pp [285]

[18] J.L.A. Hughes, E.J. McCluskey, and D.J. Lu, *Design of totally self-checking comparators with an arbitrary number of inputs*, IEEE Trans. Comput., vol. c-33, No. 6, June 1984, pp [546 - 550]

[19] H. Itoh and M. Nakumichi, *Self-checking checker designs for various 2-rail codes*, Trans. IECE of Japan, vol. E65, No. 11, November 1982, pp [665 - 671]

[20] J.C. Lo and S. Thanawastien, *The design of fast totally self-checking Berger code checkers based on Berger code partitioning*, IEEE Trans. Comput., vol. c-37, No. 2, February 1988, pp [226 - 231]

[21] J.C. Lo, and S. Thanawastien, *On the design of combinational totally self-checking 1-out-of-3 code checkers*, IEEE Trans. Comput., vol. C-39, March 1990, pp [387 - 393]

[22] M.A. Marouf and A.D. Friedman, *Efficient design of self-checking checker for m-out-of-n code*, IEEE Trans. Comput., vol. c-27, No. 6, June 1978, pp [482 - 490]

[23] M.A. Marouf and A.D, Friedman, *Design of self-checking checkers for Berger codes*, Proc. Int. Symp. Fault-tolerant computing, June 1978, pp [179 - 184]

[24] D. Nikolos, A.M. Paschalis and G. Philokyprou, *Efficient design of totally self-checking checkers for all low-cost arithmetic codes*, IEEE Trans. Comput., vol. c-37, No. 7, July 1988, pp. [808 - 814]

[25] A.M. Paschalis, C. Efslathiouf, and C. Halatsis, *An Efficient TSC 1-out-of-3 Code Checker*, IEEE Trans. Comput., vol. C-39, March 1990, pp [407 - 411]

[26] S.J. Piestrak, *Design of fast self-testing checkers for a class of Berger codes*, IEEE Trans. Comput., vol. c-36, No. 5, May 1987, pp [629 - 633]

[27] S.J. Piestrak, *Design of high-speed and cost-effective self-testing checkers for low-cost arithmetic codes*, IEEE Trans. Comput., vol. c-39, No. 3, March 1990, pp [360 - 374]

[28] D.K. Pradhan, *Error-correcting codes and self-checking circuits*, IEEE Computer, March 1980, pp [27 - 36]

[29] S.M. Reddy, *A note on self-checking checkers*, IEEE Trans. Comput., vol. C-23, October 1974, pp [1100 - 1102]

[30] D. Tao, P. Lala, and C. Hartmann, *A MOS implementation of totally self-checking checker for 1-out-of-3 code*, IEEE J. Solid-State Circuits, vol. 23, June 1988, pp [875 - 877]

[31] F.F. Sellers, M.Y. Hsiao, and L.W. Bearnson, *Error detecting logic for digital computers*, McGraw-Hill, New York, 1968

[32] J.F. Meyer and J.C. Rault, *Fault-tolerant computing: An introduction*, IEEE Trans. Comput., vol. C-25, No. 6, June 1976, pp [553 - 556]

[33] H.H. Goldstine, *The computer from Pascal to von Neumann*, Princeton, NJ: Princeton University Press, 1972

[34] J. von Neumann, *Prababilistics logics and the synthesis of reliable organisms from unreliable components*, Automata Studies, Ann, Math. Studies, No. 34. Princeton, NJ: Princeton Univ. Press, 1956, pp [43 - 98]

[35] C.V. Ramamoorthy, *Fault-tolerant computing: An introduction and an overview*, IEEE Trans. Comput., vol. c-20, No. 11, November 1971, pp [1241 - 1244]

[36] V.P. Nelson, *Fault-tolerant computing: Fundamental concepts*, IEEE Computer, July 1990, pp [19 - 25]

[37] E. Fujiwara and D.K. Pradhan, *Error-control coding in computers*, IEEE Computer, July 1990, pp [63 - 72]

[38] D.A. Rennels, *Fault-tolerant computing: Concepts and examples*, IEEE Trans. Comput., vol. c-33, No. 12, December 1984, pp [1116 - 1129]

[39] D.A. Rennels, *Distributed fault-tolerant computer systems*, IEEE Computer, October 1980, pp [55 - 64]

[40] W.C. Carter, G.R. Putzolu, A.B. Wadia, W.G. Vouricius, D.C. Jessep, E.P. Hsieh, and C.J. Tan, *Cost effectiveness of self-checking computer design*, in Dig. 7th Int. Symp. Fault-Tolerant Computing, Los Angeles, CA, June 1977, pp [117 - 123]

[41] W.C. Carter, D.C. Jessep, and A. Wadia, *Error-free decoding for failure-tolerant memories*, Proc. IEEE Int. Comput. Group Conf., June 1970, pp [229 - 239]

[42] E. Fujiwara and K. Matsuoka, *A self-checking generalized prediction checker and its use for built-in testing*, IEEE Trans. Comput., C-36, January 1987, pp [86 - 93]

[43] Z.J. Jiang and R. Venkatesan, *A new class of multi-function two-rail totally self-checking basic building blocks*, CCECE'91 Proceedings, vol.2, September 1991, pp [47.2.1 - 47.2.4]

Bibliography

B1 C.E. Allen, *Design of digital memories that tolerant all classes of defects*, Stanford University Rep. SU-SEL-66-031, Stanford University, Stanford, June 1966

B2 D.B. Armstrong, *On finding a nearly minimal set of fault detection tests for combinational logic networks*, IEEE Trans. Electron. Comput., vol. EC-15, February 1966, pp [66 - 73]

B3 A. Avizienis, *Fault-tolerant computing: An overview*, IEEE Computer, vol. 4, Jan./Feb. 1971, pp [5 - 8]

B4 D.C. Bossen, *B-adjacent error correction*, IBM J. Res. Develop., vol. 14, 1970, pp [402 - 408]

B5 W.C. Carter, *Fault-tolerant computing: An introduction and a viewpoint*, IEEE Trans. Comput., vol. c-22, No. 3, March 1973, pp [225 - 229]

B6 W.C. Carter and C.E. McCarthy, *Implementation of an experimental fault-tolerant memory system*, IEEE Trans. Comput., vol. C-25, No. 6, June 1976, pp [557 - 568]

B7 W.C. Carter, G.R. Putzolu, A.B. Wadia, W.G. Bouricius, D.C. Jessep, E.P. Hsieh, and C.J. Tan, *Cost effectiveness of self-checking computer design*, Dig.,

7th Annu. Int. Symp. Fault-Tolerant Computing, Los Angeles, CA, June 23-30 1977, pp [117 - 123]

B8 J. Chavade and Y. Crouzet, *P.A.D.: A self-checking LSI circuit for fault-detection in microcomputers*, Dig. 12th Annu. Int. Symp. Fault-Tolerant Comput., Santa Monica, CA, June 22-24, 1982, pp [55 - 62]

B9 Y. Crouzet and C. Landrault, *Design of self-checking MOS-LSI circuits, application to a four-bit microprocessor*, Dig. 9th Annu. Int. Symp. Fault-Tolerant Comput., Madison, Wis., June 20-22, 1979, pp [189 - 192]

B10 Y. Crouzet and C. Landrault, *Design of specification of a self-checking detection processor*, Dig. 10th Annu. Int. Symp. Fault-Tolerant Comput., Kyoto, Japan, October 1-3, 1980, pp [275 - 277]

B11 M. Diaz, *design of totally self-checking and fail safe sequential machines*, in Dig. 4th Annu. Symp. on Fault-Tolerant Computing, June 1974, pp [(3-19) - (3-24)]

B12 N. Gaitanis, *A totally self-checking error indicator*, IEEE Trans. Comput., vol. C-34, No. 8, August 1985, pp [758 - 761]

B13 H. Ito, *A 2-rail logic combinational circuit with easy detection of stuck-open and stuck-on faults in FETs*, in Pacific Rim International Symposium on Fault Tolerant System, September 1991, Kawasaki, Japan, pp [252 - 257]

B14 J.P. Khakbaz and E. J. McCluskey, *Self-testing embedded parity checkers exaustive XOR gate testing*, Stanford University, CRC ReP. 82-10/CSL TN207, June 1982

B15 J.P. Khakbaz, *Totally self-checking checker for 1-out-of-n code using two-rail codes*, IEEE Trans. Comput., vol. c-31, No. 7, July 1982, pp [677 - 681]

B16 C.R. Kime, *Fault-tolerant computing: An introduction and a perspective*, IEEE Trans. Comput., vol. C-24, No. 5, May 1975, pp [457 - 460]

B17 J. Losq, *A highly efficient redundancy scheme: Self-purging redundancy*, IEEE Trans. Comput., vol. C-25, No. 6, June 1976, pp [637 - 642]

B18 J.P. Mark, J.A. Abraham, and E.S. Davidson, *The design of PLAs with concurrent error detection*, in Proc. 12th FTCS, Santa Monica, CA, June 22-24, 1982

B19 E.J. McCluskey and F.W. Clegg, *Fault equivalence in combinational logic networks*, IEEE Trans. Comput., vol. C-20, November 1971, pp [1286 - 1293]

B20 E.J. McCluskey, *Design techniques for testable embedded error checkers*, IEEE Computer, July 1990, pp [84 - 87]

B21 K.C.Y. Mei, *Bridging and stuck-at faults*, IEEE Trans. Comput., vol. C-23, July 1974, pp [720 - 727]

B22 M. Nicolaidis and B. Courtois, *Strongly code disjoint checkers*, IEEE Trans. Comput., vol. 37, No. 6, June 1988, pp [751 - 756]

B23 F. Ozguner, *Design of totally self-checking asynchronous and synchronous sequential machines*, in Proc. 7th Annu. Int. Conf. on Fault-Tolerant Computing, June 1977, pp [124 - 129]

B24 D.A. Rennels, A. Avizienis, and M. Ercegovac, *A study of standard building blocks for the design of fault-tolerant distrbuted computer systems*, Dig., 8th

Annu. Int. Symp. fault-Tolerant Computing, Toulouse, France, June 21-23, 1978, pp [144 - 149]

B25 D.R. Schertz, *Fault-tolerant computing: An introduction*, IEEE Trans. Comput., vol. C-23, No. 7, July 1974, pp [649 - 650]

B26 J.E. Smith, *The design of totally self-checking combinational circuits*, Coordinated Science Laboratory Rep. R-737, University of Illinois, Urbana, August 1976

B27 J.E. Smith and G. Metze, *The design of totally self-checking combinational circuits*, Dig. 7th Annu. Int. Symp. Fault-Tolerant Comput., Los Angeles, June 28-30, 1977, pp [130 - 134]

B28 J.E. Smith *The design of totally self-checking check circuits for a class of unordered codes*, J. Design Automation and Fault-Tolerant Comput., vol. 1, October 1977, pp [321 - 342]

B29 J.E. Smith and G. Metze, *Strongly fault secure logic networks*, IEEE Trans. Comput., vol. C-27, No. 6, June 1978, pp [491 - 499]

B30 J.G. Tryon, *Redundancy Techniques for computing systems*, Wilcox and Mann, Eds. Washington, DC: Spartan Books, 1962, pp [205 - 228]

B31 J. Viaud and R. David, *Sequentially self-checking circuits*, in Proc. 10th FTCS, June 1980, pp [263 - 268]

B32 J.F. Wakerly, *Partially self-checking circuits and their use in performing logical operations*, IEEE Trans. Comput., vol. C-23, July 1974, pp [658 - 666]

B33 J.F. Wakerly, *Checked binary addition with checksum codes*, J. Design Automation and Fault-Tolerant Computing, vol. 1, October 1976, pp [18 - 27]

B34 S.B. William, *Bell Telephone Laboratories relay computing system*, Ann. Computation Lab., Harvard Univ., vol. XVI, 1948, pp [41 - 53]

Appendix A

Verifications of TSC property for the proposed EIS BBBs

Table A.1: The truth table of EIS-XOR BBB which has a stuck-at fault at gate-1.

Input **A**		Input **B**		Stuck-at-0		Stuck-at-1	
A_1	A_2	B_1	B_2	E_1	E_2	E_1	E_2
0	1	0	1	0	1	0	1
0	1	1	0	1	0	1	1
1	0	0	1	1	0	1	1
1	0	1	0	0	0	0	1

Table A.2: The truth table of EIS-XOR BBB which has a stuck-at fault at gate-2.

Input **A**		Input **B**		Stuck-at-0		Stuck-at-1	
A_1	A_2	B_1	B_2	E_1	E_2	E_1	E_2
0	1	0	1	0	0	0	1
0	1	1	0	1	0	1	1
1	0	0	1	1	0	1	1
1	0	1	0	0	1	0	1

Table A.3: The truth table of EIS-XOR BBB which has a stuck-at fault at gate-3.

Input **A**		Input **B**		Stuck-at-0		Stuck-at-1	
A_1	A_2	B_1	B_2	E_1	E_2	E_1	E_2
0	1	0	1	0	1	1	1
0	1	1	0	1	0	1	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	1	1

Table A.4: The truth table of EIS-XOR BBB which has a stuck-at fault at gate-4.

Input **A**		Input **B**		Stuck-at-0		Stuck-at-1	
A_1	A_2	B_1	B_2	E_1	E_2	E_1	E_2
0	1	0	1	0	1	1	1
0	1	1	0	0	0	1	0
1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	1

Table A.5: The truth table of EIS-XOR BBB which has a stuck-at fault at gate-5.

Input A		Input B		Stuck-at-0		Stuck-at-1	
A_1	A_2	B_1	B_2	E_1	E_2	E_1	E_2
0	1	0	1	0	0	0	1
0	1	1	0	1	0	1	1
1	0	0	1	1	0	1	1
1	0	1	0	0	0	0	1

Table A.6: The truth table of EIS-XOR BBB which has a stuck-at fault at gate-6.

Input A		Input B		Stuck-at-0		Stuck-at-1	
A_1	A_2	B_1	B_2	E_1	E_2	E_1	E_2
0	1	0	1	0	1	1	1
0	1	1	0	0	0	1	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	1	1

Table A.7: The truth table of EIS-AND BBB which has a stuck-at fault at gate 1.

Input A		Input B		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	1	1	1
1	0	0	1	1	0	0	1	1	1	1	1
1	0	1	0	0	0	0	0	0	1	1	0

Table A.8: The truth table of EIS-AND BBB which has a stuck-at fault at gate-2.

Input A		Input B		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	0	1	1	0	1	0	1
0	1	1	0	1	0	0	1	1	1	1	1
1	0	0	1	1	0	0	1	1	1	1	1
1	0	1	0	0	1	1	0	0	1	1	0

Table A.9: The truth table of EIS-AND BBB which has a stuck-at fault at gate-3.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	1	1	1	0	0
0	1	1	0	1	0	0	1	1	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	0	1	1	0	1	1	1	1

Table A.10: The truth table of EIS-AND BBB which has a stuck-at fault at gate 1

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	1	1	1	0	0
0	1	1	0	0	0	0	0	1	0	0	1
1	0	0	1	1	0	0	1	1	0	0	1
1	0	1	0	0	1	1	0	1	1	1	1

Table A.11: The truth table of EIS-AND BBB which has a stuck-at fault at gate 5.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	0	1	1	0	1	0	1
0	1	1	0	1	0	0	1	1	1	1	1
1	0	0	1	1	0	0	1	1	1	1	1
1	0	1	0	0	0	0	0	0	1	1	0

Table A.12: The truth table of EIS-AND BBB which has a stuck-at fault at gate-6.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	1	1	1	0	0
0	1	1	0	0	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	0	1	1	0	1	1	1	1

Table A.13: The truth table of EIS-AND BBB which has a stuck-at fault at gate-7.

Input A		Input B		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	1	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	1
1	0	0	1	1	0	0	1	1	0	1	1
1	0	1	0	0	1	1	0	0	1	0	0

Table A.14: The truth table of EIS-AND BBB which has a stuck-at fault at gate-8.

Input A		Input B		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	0	0	1	0	1
0	1	1	0	1	0	0	1	1	0	0	0
1	0	0	1	1	0	0	1	1	0	0	0
1	0	1	0	0	1	1	0	0	1	1	1

Table A.15: The truth table of EIS-AND BBB which has a stuck-at fault at gate-9.

Input A		Input B		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	1	0	1	1	1
0	1	1	0	1	0	0	1	1	0	1	1
1	0	0	1	1	0	0	1	1	0	1	1
1	0	1	0	0	1	0	0	0	1	1	0

Table A.16: The truth table of EIS-AND BBB which has a stuck-at fault at gate-10.

Input A		Input B		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	0	0	1	0	1
0	1	1	0	1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0	1	0	0	1
1	0	1	0	0	1	1	0	0	1	1	1

Table A.17: The truth table of EIS-OR BBB which has a stuck-at fault at gate-1.

Input A		Input B		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	1	0	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1
1	0	1	0	0	0	1	1	0	1	1	0

Table A.18: The truth table of EIS-OR BBB which has a stuck-at fault at gate-2.

Input A		Input B		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	0	0	0	0	1	0	1
0	1	1	0	1	0	1	0	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1
1	0	1	0	0	1	1	0	0	1	1	0

Table A.19: The truth table of EIS-OR BBB which has a stuck-at fault at gate 3.

Input A		Input B		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	1	1	1	1
0	1	1	0	1	0	1	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0
1	0	1	0	0	1	1	0	1	1	0	0

Table A.20: The truth table of EIS-OR BBB which has a stuck-at fault at gate-1.

Input A		Input B		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	1	1	1	1
0	1	1	0	0	0	0	0	1	0	1	0
1	0	0	1	1	0	1	0	1	0	1	0
1	0	1	0	0	1	1	0	1	1	0	0

Table A.21: The truth table of EIS-OR BBB which has a stuck-at fault at gate-5.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	0	0	0	0	1	0	1
0	1	1	0	1	0	1	0	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1
1	0	1	0	0	0	1	1	0	1	1	0

Table A.22: The truth table of EIS-OR BBB which has a stuck-at fault at gate-6.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	1	1	1	1
0	1	1	0	0	0	0	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0
1	0	1	0	0	1	1	0	1	1	0	0

Table A.23: The truth table of EIS-OR BBB which has a stuck-at fault at gate-7.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	1	1
0	1	1	0	1	0	1	0	1	0	0	0
1	0	0	1	1	0	1	0	1	0	0	0
1	0	1	0	0	1	0	0	0	1	1	0

Table A.24: The truth table of EIS-OR BBB which has a stuck-at fault at gate-8.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0	1	0	1	1
1	0	0	1	1	0	1	0	1	0	1	1
1	0	1	0	0	1	1	1	0	1	1	0

Table A.25: The truth table of EIS-OR BBB which has a stuck at fault at gate 9.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	1	1
0	1	1	0	1	0	0	0	1	0	1	0
1	0	0	1	1	0	0	0	1	0	1	0
1	0	1	0	0	1	0	0	0	1	1	0

Table A.26: The truth table of EIS-OR BBB which has a stuck at fault at gate 10.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	0	0	1	0	1
0	1	1	0	1	0	1	0	1	0	1	1
1	0	0	1	1	0	1	0	1	0	1	1
1	0	1	0	0	1	1	0	0	1	1	1

Appendix B

Verifications of TSC property for the proposed EISS BBBs

Table B.1: The truth table of EISS-XOR BBB which has a stuck-at fault at gate-1.

Input **A**		Input **B**		Stuck-at-0		Stuck-at-1	
A_1	A_2	B_1	B_2	E_1	E_2	E_1	E_2
0	1	0	1	0	1	1	1
0	1	1	0	0	0	1	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	1	1

Table B.2: The truth table of EISS-XOR BBB which has a stuck-at fault at gate-2.

Input **A**		Input **B**		Stuck-at-0		Stuck-at-1	
A_1	A_2	B_1	B_2	E_1	E_2	E_1	E_2
0	1	0	1	0	0	0	1
0	1	1	0	1	0	1	1
1	0	0	1	1	0	1	1
1	0	1	0	0	0	0	1

Table B.3: The truth table of EISS-AND BBB which has a stuck-at fault at gate-1.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	0	1	1	0	1	0	1
0	1	1	0	1	0	0	1	1	1	1	1
1	0	0	1	1	0	0	1	1	1	1	1
1	0	1	0	0	0	0	0	0	1	1	0

Table B.4: The truth table of EISS-AND BBB which has a stuck-at fault at gate-2.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	1	1	1	0	0
0	1	1	0	0	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	0	1	1	0	1	1	1	1

Table B.5: The truth table of EISS-AND BBB which has a stuck-at fault at gate-3.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	1	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	1
1	0	0	1	1	0	0	1	1	0	1	1
1	0	1	0	0	1	1	0	0	1	0	0

Table B.6: The truth table of EISS-AND BBB which has a stuck-at fault at gate-4.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	0	0	1	0	1
0	1	1	0	1	0	0	1	1	0	0	0
1	0	0	1	1	0	0	1	1	0	0	0
1	0	1	0	0	1	1	0	0	1	1	1

Table B.7: The truth table of EISS-AND BBB which has a stuck-at fault at gate-5.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	0	1	0	1	0	1	1	1
0	1	1	0	1	0	0	1	1	0	1	1
1	0	0	1	1	0	0	1	1	0	1	1
1	0	1	0	0	1	0	0	0	1	1	0

Table B.8: The truth table of EISS-AND BBB which has a stuck-at fault at gate-6.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	E_1	E_2	C_1	C_2
0	1	0	1	1	0	0	0	0	1	0	1
0	1	1	0	0	1	0	0	1	0	0	1
1	0	0	1	0	1	0	0	1	0	0	1
1	0	1	0	1	0	1	0	0	1	1	1

Table B.9: The truth table of EISS-OR BBB which has a stuck-at fault at gate-1.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	1	1	1	1
0	1	1	0	0	C	0	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0
1	0	1	0	0	1	1	0	1	1	0	0

Table B.10: The truth table of EISS-OR BBB which has a stuck-at fault at gate-2.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	0	0	0	0	1	0	1
0	1	1	0	1	0	1	0	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1
1	0	1	0	0	0	1	1	0	1	1	0

Table B.11: The truth table of EISS-OR BBB which has a stuck-at fault at gate-3.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	1	1
0	1	1	0	1	0	1	0	1	0	0	0
1	0	0	1	1	0	1	0	1	0	0	0
1	0	1	0	0	1	0	0	0	1	1	0

Table B.12: The truth table of EISS-OR BBB which has a stuck-at fault at gate-4.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0	1	0	1	1
1	0	0	1	1	0	1	0	1	0	1	1
1	0	1	0	0	1	1	1	0	1	1	0

Table B.13: The truth table of EISS-OR BBB which has a stuck-at fault at gate-5.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	1	1
0	1	1	0	1	0	0	0	1	0	1	0
1	0	0	1	1	0	0	0	1	0	1	0
1	0	1	0	0	1	0	0	0	1	1	0

Table B.14: The truth table of EISS-OR BBB which has a stuck-at fault at gate-6.

Input **A**		Input **B**		Stuck-at-0				Stuck-at-1			
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	E_1	E_2	D_1	D_2
0	1	0	1	0	1	0	0	0	1	0	1
0	1	1	0	1	0	1	0	1	0	1	1
1	0	0	1	1	0	1	0	1	0	1	1
1	0	1	0	0	1	1	0	0	1	1	1

Appendix C

Verifications of TSC property for the proposed EIIS BBBs

Table C.1: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-1.

Input A		Input B		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	1	1	1	1	0	1	0
1	0	0	1	1	1	1	1	1	0	1	0
1	0	1	0	0	1	1	0	1	0	0	1

Table C.2: The truth table of EHS-XOR BBB which has a stuck-at-1 fault at gate-1.

Input A		Input B		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	0	1	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	0	0	0	1	0	0	1

Table C.3: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-2.

Input A		Input B		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	0	0	0	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0
1	0	1	0	0	1	1	0	1	0	0	1

Table C.4: The truth table of EHS-XOR BBB which has a stuck-at-1 fault at gate-2.

Input A		Input B		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	1	1	0	0	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	1	1	1	1	1	0	0	1

Table C.5: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-3.

Input **A**		Input **B**		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	1	1	1	1	1	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0	1	0	0	1

Table C.6: The truth table of EHS-XOR BBB which has a stuck-at-1 fault at gate-3.

Input **A**		Input **B**		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	1	1	0	0	0	0
1	0	0	1	1	0	1	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	1	1

Table C.7: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-4.

Input **A**		Input **B**		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	0	0	0	0	0
0	1	1	0	1	0	0	1	1	1	1	1
1	0	0	1	1	0	0	1	1	1	1	1
1	0	1	0	0	1	1	0	1	0	0	1

Table C.8: The truth table of EHS-XOR BBB which has a stuck-at-1 fault at gate-4.

Input **A**		Input **B**		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	0	1	1	1	1
1	0	0	1	1	0	0	0	1	1	1	1
1	0	1	0	0	1	1	1	1	1	0	0

Table C.9: The truth table of EIIS-XOR BBB which has a stuck-at-0 fault at gate-5.

Input **A**		Input **B**		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	0	0	0	0	1	1

Table C.10: The truth table of EIIS-XOR BBB which has a stuck-at-1 fault at gate-5.

Input **A**		Input **B**		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	1	1	1	1	1	1
0	1	1	0	1	0	1	1	0	0	0	0
1	0	0	1	1	0	1	1	0	0	0	0
1	0	1	0	0	1	1	0	1	0	0	1

Table C.11: The truth table of EIIS-XOR BBB which has a stuck-at-0 fault at gate-6.

Input **A**		Input **B**		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	0	0	0	0	0
0	1	1	0	1	0	0	0	1	1	1	1
1	0	0	1	1	0	0	0	1	1	1	1
1	0	1	0	0	1	1	0	1	0	0	1

Table C.12: The truth table of EIIS-XOR BBB which has a stuck-at-1 fault at gate-6.

Input **A**		Input **B**		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	1	1	1	0	0

Table C.13: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-7.

Input **A**		Input **B**		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	0	0	0	0
1	0	0	1	1	0	0	1	0	0	0	0
1	0	1	0	0	1	1	0	1	0	0	1

Table C.14: The truth table of EHS-XOR BBB which has a stuck-at-1 fault at gate-7.

Input **A**		Input **B**		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	1	1	1	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0	1	0	0	1

Table C.15: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-8.

Input **A**		Input **B**		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	0	0	0
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0	1	0	0	1

Table C.16: The truth table of EHS-XOR BBB which has a stuck-at-1 fault at gate-8.

Input **A**		Input **B**		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	1	1	1
1	0	0	1	1	0	0	1	1	1	1	1
1	0	1	0	0	1	1	0	1	1	0	0

Table C.17: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-9.

Input A		Input B		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	0	0	1	1	1	1	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	0	1	0	0	0	1	1

Table C.18: The truth table of EHS-XOR BBB which has a stuck-at-1 fault at gate-9.

Input A		Input B		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	1	1	1	0	0	0	0
1	0	0	1	1	1	1	1	0	0	0	0
1	0	1	0	0	1	1	0	1	0	0	1

Table C.19: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-10.

Input A		Input B		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0	1	0	1	1

Table C.20: The truth table of EHS-XOR BBB which has a stuck at-1 fault at gate-10.

Input A		Input B		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	1	1
0	1	1	0	1	0	0	1	1	0	0	0
1	0	0	1	1	0	0	1	1	0	0	0
1	0	1	0	0	1	1	0	1	0	0	1

Table C.21: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-11.

Input **A**		Input **B**		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0	1	0	0	0

Table C.22: The truth table of EHS-XOR BBB which has a stuck-at-1 fault at gate-11.

Input **A**		Input **B**		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	0
0	1	1	0	1	0	0	1	1	0	1	1
1	0	0	1	1	0	0	1	1	0	1	1
1	0	1	0	0	1	1	0	1	0	0	1

Table C.23: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-12.

Input **A**		Input **B**		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	0	0
1	0	0	1	1	0	0	1	1	0	0	0
1	0	1	0	0	1	1	0	1	0	0	1

Table C.24: The truth table of EHS-XOR BBB which has a stuck-at-1 fault at gate-12.

Input **A**		Input **B**		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	1	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0	1	0	1	1

Table C.25: The truth table of EHS-XOR BBB which has a stuck-at-0 fault at gate-13.

Input A		Input B		Stuck-at-0							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	0
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0	1	0	0	0

Table C.26: The truth table of EHS-XOR BBB which has a stuck-at 1 fault at gate-13.

Input A		Input B		Stuck-at-1							
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1^*	E_2^*
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	1
1	0	0	1	1	0	0	1	1	0	1	1
1	0	1	0	0	1	1	0	1	0	0	1

Table C.27: The truth table of EHS-AND BBB which has a stuck-at fault at gate-1.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	C_1	C_2	E_1	E_2	D_1	D_2	C_1	C_2
0	1	0	1	0	1	0	1	0	1	1	1	1	1	0	1
0	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1
1	0	0	0	0	1	0	0	0	1	1	0	1	0	0	1
1	0	0	1	0	0	1	0	1	0	1	1	0	0	1	0

Table C.28: The truth table of EHS-AND BBB which has a stuck-at fault at gate-2.

Input A		Input B		Stuck-at-0						Stuck at-1					
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	C_1	C_2	E_1	E_2	D_1	D_2	C_1	C_2
0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1
0	1	1	0	1	1	1	1	0	1	1	0	1	0	0	1
1	0	0	1	1	1	1	1	0	1	1	0	1	0	0	1
1	0	1	0	0	1	1	0	1	0	1	0	0	1	1	0

Table C.29: The truth table of EHS-AND BBB which has a stuck-at fault at gate-3.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	C_1	C_2	E_1	E_2	D_1	D_2	C_1	C_2
0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1
0	1	1	0	1	0	1	0	0	1	1	0	0	0	1	1
1	0	0	1	1	0	1	0	0	1	1	0	0	0	1	1
1	0	1	0	0	1	0	0	0	0	0	1	1	0	1	0

Table C.30: The truth table of EHS-AND BBB which has a stuck-at fault at gate-1.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	C_1	C_2	E_1	E_2	D_1	D_2	C_1	C_2
0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0
0	1	1	0	1	0	1	0	0	1	1	0	1	1	0	0
1	0	0	1	1	0	1	0	0	1	1	0	1	1	0	0
1	0	1	0	0	1	1	1	1	1	0	1	1	0	1	0

Table C.31: The truth table of EHS-AND BBB which has a stuck-at fault at gate-5.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	C_1	C_2	E_1	E_2	D_1	D_2	C_1	C_2
0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1
0	1	1	0	1	0	0	0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1
1	0	1	0	0	1	0	0	0	0	0	1	1	0	1	0

Table C.32: The truth table of EHS-AND BBB which has a stuck-at fault at gate-6.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	C_1	C_2	E_1	E_2	D_1	D_2	C_1	C_2
0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	1
0	1	1	0	1	0	1	0	0	1	1	0	1	1	0	0
1	0	0	1	1	0	1	0	0	1	1	0	1	1	0	0
1	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1

Table C.33: The truth table of EHS-AND BBB which has a stuck-at fault at gate-7.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	C_1	C_2	E_1	E_2	D_1	D_2	C_1	C_2
0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1
0	1	1	0	1	0	1	0	0	1	1	0	1	0	1	1
1	0	0	1	1	0	1	0	0	1	1	0	1	0	1	1
1	0	1	0	0	1	1	0	0	0	0	1	1	0	1	0

Table C.34: The truth table of EHS-AND BBB which has a stuck-at fault at gate 8.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	C_1	C_2	E_1	E_2	D_1	D_2	C_1	C_2
0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1
0	1	1	0	1	0	1	0	0	0	1	0	1	0	0	1
1	0	0	1	1	0	1	0	0	0	1	0	1	0	0	1
1	0	1	0	0	1	1	0	1	0	0	1	1	0	1	1

Table C.35: The truth table of EHS-AND BBB which has a stuck-at fault at gate 9.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	D_1	D_2	C_1	C_2	E_1	E_2	D_1	D_2	C_1	C_2
0	1	0	1	0	0	0	0	0	0	0	1	0	1	0	1
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	0
1	0	0	1	1	0	1	0	0	1	1	1	1	1	0	0
1	0	1	0	0	0	1	1	1	1	0	1	1	0	1	0

Table C.36: The truth table of EHS-OR BBB which has a stuck-at fault at gate-1.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1	E_2	C_1	C_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	1
0	1	0	0	1	1	1	1	1	0	1	0	0	1	1	0
1	0	0	0	1	1	1	1	1	0	1	0	0	1	1	0
1	0	0	1	0	1	1	0	1	0	0	0	0	0	1	0

Table C.37: The truth table of EHS-OR BBB which has a stuck-at fault at gate-2.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1	E_2	C_1	C_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	1	1	0	0	0	1
0	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0
1	0	0	1	0	0	0	0	1	0	1	0	0	1	1	0
1	0	1	0	0	1	1	0	1	0	1	1	1	1	1	0

Table C.38: The truth table of EHS-OR BBB which has a stuck-at fault at gate-3.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1	E_2	C_1	C_2	D_1	D_2
0	1	0	1	0	1	1	1	1	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	0	1	1	0	0
1	0	0	1	1	0	0	1	1	0	1	0	1	1	0	0
1	0	1	0	0	1	1	0	1	0	0	1	0	0	0	0

Table C.39: The truth table of EHS-OR BBB which has a stuck-at fault at gate-4.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1	E_2	C_1	C_2	D_1	D_2
0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	1	1	0	0	0	1	1
1	0	0	1	1	0	0	1	1	1	1	0	0	0	1	1
1	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1

Table C.40: The truth table of EHS-OR BBB which has a stuck-at fault at gate-5.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1	E_2	C_1	C_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1
0	1	1	0	1	0	0	1	1	0	1	0	1	1	0	0
1	0	0	1	1	0	0	1	1	0	1	0	1	1	0	0
1	0	1	0	0	1	0	0	0	0	0	1	1	0	1	0

Table C.41: The truth table of EHS-OR BBB which has a stuck-at fault at gate-6.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1	E_2	C_1	C_2	D_1	D_2
0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	1
0	1	1	0	1	0	0	0	1	1	1	0	0	1	1	0
1	0	0	1	1	0	0	0	1	1	1	0	0	1	1	0
1	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1

Table C.42: The truth table of EHS-OR BBB which has a stuck-at fault at gate-7.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1	E_2	C_1	C_2	D_1	D_2
0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1
0	1	1	0	1	0	0	1	0	0	1	0	0	1	1	0
1	0	0	1	1	0	0	1	0	0	1	0	0	1	1	0
1	0	1	0	0	1	1	0	0	0	0	1	1	0	1	0

Table C.43: The truth table of EHS-OR BBB which has a stuck-at fault at gate-8.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1	E_2	C_1	C_2	D_1	D_2
0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	0	0	1	1	1
1	0	0	1	1	0	0	1	1	0	1	0	0	1	1	1
1	0	1	0	0	1	1	0	1	0	0	1	1	0	1	1

Table C.44: The truth table of EHS-OR BBB which has a stuck-at fault at gate-9.

Input A		Input B		Stuck-at-0						Stuck-at-1					
A_1	A_2	B_1	B_2	E_1	E_2	C_1	C_2	D_1	D_2	E_1	E_2	C_1	C_2	D_1	D_2
0	1	0	1	0	0	1	1	1	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1	1	0	1	1	1	1	0	0
1	0	0	1	1	0	0	1	1	0	1	1	1	1	0	0
1	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0

Appendix D

Verifications of TSC property for the proposed MF BBBs

Table D.1: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-1.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR\ E'$		AND C		OR D		XOR E		$\overline{XOR}\ E^*$		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	1	1	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	1

Table D.2: The truth table of MF-OR BBB which has a stuck-at 0 fault at gate-1.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR\ E'$		AND C		OR D		XOR E		$\overline{XOR}\ E^*$		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	0	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.3: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-2.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR\ E'$		AND C		OR D		XOR E		$\overline{XOR}\ E^*$		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	1	1	1	1	0	0	0	1	1	0	1	1	1	1	1

Table D.4: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-2.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.5: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-3.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	1	1	1	1	0	0	0	1	1	0	1	1	1	1	1

Table D.6: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-3.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.7: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-4.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	1	1	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	1

Table D.8: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-4.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.9: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-5.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.10: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-5.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	0	1	1	1	1	1	0	0	0	1	1	0	1	1	1	1	1

Table D.11: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-6.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	1	1	1	0	0	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	0	0	1	1	1	0	0

Table D.12: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-6.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.13: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-7.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	1	1	1	0	0	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	0	0	1	1	1	0	0

Table D.14: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-7.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.15: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-8.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR$ E'		AND C		OR D		XOR E		XOR E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	1	0	0	0	0	0	1	0	0	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	1	0	1	1	1	1	1

Table D.16: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-8.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR$ E'		AND C		OR D		XOR E		XOR E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.17: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-9.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR$ E'		AND C		OR D		XOR E		XOR E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	1	1	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	1

Table D.18: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-9.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR$ E'		AND C		OR D		XOR E		XOR E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	0	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.19: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-10.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.20: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-10.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	0	1	1	1	1	1	0	0	0	1	1	0	1	1	1	1	1

Table D.21: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-11.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	1	1	1	1	0	0	0	1	1	0	1	1	1	1	1

Table D.22: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-11.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.23: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-12.

Input A		Input B		MAND C'		MOR D'		MCOR F'		MXOR E'		AND C		OR D		XOR E		XOR E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	1	1	1	0	0	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	0	0	1	1	1	0	0

Table D.24: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-12.

Input A		Input B		MAND C'		MOR D'		MCOR F'		MXOR E'		AND C		OR D		XOR E		XOR E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	0	0	1	1	1	0	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.25: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-13.

Input A		Input B		MAND C'		MOR D'		MCOR F'		MXOR E'		AND C		OR D		XOR E		XOR E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	1

Table D.26: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-13.

Input A		Input B		MAND C'		MOR D'		MCOR F'		MXOR E'		AND C		OR D		XOR E		XOR E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.27: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-14.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.28: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-14.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	1	1	1	1	0	0	0	1	1	0	1	1	1	1	1

Table D.29: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-15.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	0	0	0	0	1	0	1	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.30: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-15.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	1	1	1	1	0	0	1	1	1	0	1	1	1	1	1

Table D.31: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-16.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	1	1	1	1	0	0	1	1	1	0	1	1	1	1	1

Table D.32: The truth table of MF-OR BBB which has a stuck-at 0 fault at gate 16.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	0	0	0	0	1	0	1	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.33: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-17.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	1	1	0	1	0	1	0	1	0	0	0	1	1	1	0	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	0	0	1	0	1	0	0	1	0	0	0	1	1	1	0	0

Table D.34: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-17.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	1	1	0	1	0	1	0	1	1	1	1	0	0	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.35: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-18.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	1	1	0	1	0	1	0	1	0	0	0	1	1	1	0	0
0	1	0	1	1	0	1	1	0	1	0	1	0	1	1	1	1	0	0	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.36: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-18.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	0	0	1	0	1	0	0	1	0	0	0	1	1	1	0	0

Table D.37: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-19.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	1	1	1	0	0	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.38: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-19.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	1	1	0	1	0	1	0	1	0	0	0	1	1	1	0	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	0	0	1	1	1	0	0

Table D.39: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-20.

Input A		Input B		MAND C'		MOR D'		MCOR F'		MXOR E'		AND C		OR D		XOR E		XOR E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	F'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1	0	0	0	1	1
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	1	0	1	1	1	1	1
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	1	0	1	1	1	1	1

Table D.40: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-20.

Input A		Input B		MAND C'		MOR D'		MCOR F'		MXOR E'		AND C		OR D		XOR E		XOR E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	F'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.41: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-21.

Input A		Input B		MAND C'		MOR D'		MCOR F'		MXOR E'		AND C		OR D		XOR E		XOR E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	F'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	1	1	1	0	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	1	1	1	1	0	0

Table D.42: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-21.

Input A		Input B		MAND C'		MOR D'		MCOR F'		MXOR E'		AND C		OR D		XOR E		XOR E'		OR D'	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	F'_2	E'_1	D'_1	D'_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	0	0	0	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.43: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-22.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
I_1	I_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.44: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-22.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0

Table D.45: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-23.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$\overline{MXOR}$ E'		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}$ E^*		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	1	1	0	1	1	0	1	1	1	1	0	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	0	0	0	1	1	0	1	1	1	1	0	0

Table D.46: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-23.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR\ E'$		AND C		OR D		XOR E		$XOR\ E'$		OR D'	
A_1	A_2	B_1	B_2	C_1'	C_2'	D_1'	D_2'	F_1'	F_2'	E_1'	E_2'	C_1	C_2	D_1	D_2	E_2	E_1	F_2'	E_1'	D_1'	D_2'
1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	1	1	0	1	0	1	0	0	0	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.47: The truth table of MF-OR BBB which has a stuck-at 1 fault at gate-24.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR\ E'$		AND C		OR D		XOR E		$XOR\ E'$		OR D'	
A_1	A_2	B_1	B_2	C_1'	C_2'	D_1'	D_2'	F_1'	F_2'	E_1'	E_2'	C_1	C_2	D_1	D_2	E_2	E_1	F_2'	E_1'	D_1'	D_2'
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	1	1	0	1	1	0	1	1	1	1	0	0
0	1	0	1	1	0	0	1	0	1	1	1	0	1	0	1	0	0	0	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.48: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-24.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR\ E'$		AND C		OR D		XOR E		$XOR\ E'$		OR D'	
A_1	A_2	B_1	B_2	C_1'	C_2'	D_1'	D_2'	F_1'	F_2'	E_1'	E_2'	C_1	C_2	D_1	D_2	E_2	E_1	F_2'	E_1'	D_1'	D_2'
1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	0	0	0	1	1	0	1	1	1	1	1	1

Table D.49: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-25.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXOR\ E'$		AND C		OR D		XOR E		$XOR\ E'$		OR D'	
A_1	A_2	B_1	B_2	C_1'	C_2'	D_1'	D_2'	F_1'	F_2'	E_1'	E_2'	C_1	C_2	D_1	D_2	E_2	E_1	F_2'	E_1'	D_1'	D_2'
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	0	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	0	0

Table D.50: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-25.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXNOR\ E'$		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}\ E^*$		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.51: The truth table of MF-OR BBB which has a stuck-at-1 fault at gate-26.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXNOR\ E'$		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}\ E^*$		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	1	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0

Table D.52: The truth table of MF-OR BBB which has a stuck-at-0 fault at gate-26.

Input A		Input B		MAND C'		MOR D'		MCOR F'		$MXNOR\ E'$		AND C		OR D		$\overline{XOR}$ E		$\overline{XOR}\ E^*$		OR D^*	
A_1	A_2	B_1	B_2	C'_1	C'_2	D'_1	D'_2	F'_1	F'_2	E'_1	E'_2	C_1	C_2	D_1	D_2	E_2	E_1	E^*_2	E^*_1	D^*_1	D^*_2
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0
0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	0	0
0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	1	0	0

Appendix E

6-bit diagnostic sequence pairs

Table E.1: The table of 6-bit diagnostic sequence pairs ($W_1 - W_5$).

W_1		W_2		W_3		W_4		W_5	
w_1	$\overline{w_1}$	w_2	$\overline{w_2}$	w_3	$\overline{w_3}$	w_4	$\overline{w_4}$	w_5	$\overline{w_5}$
1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1
1	0	0	1	0	1	0	1	1	0
0	1	0	1	1	0	0	1	0	1
1	0	1	0	1	0	1	0	1	0

Table E.2: The table of 6-bit diagnostic sequence pairs ($W_6 - W_{10}$).

W_6		W_7		W_8		W_9		W_{10}	
w_6	$\overline{w_6}$	w_7	$\overline{w_7}$	w_8	$\overline{w_8}$	w_9	$\overline{w_9}$	w_{10}	$\overline{w_{10}}$
1	0	1	0	1	0	1	0	1	0
0	1	0	1	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	1
1	0	1	0	0	1	0	1	0	1
1	0	0	1	1	0	1	0	1	0
0	1	0	1	0	1	0	1	1	0

Table E.3: The table of 6-bit diagnostic sequence pairs ($W_{11} - W_{15}$).

W_{11}		W_{12}		W_{13}		W_{14}		W_{15}	
w_{11}	$\overline{w_{11}}$	w_{12}	$\overline{w_{12}}$	w_{13}	$\overline{w_{13}}$	w_{14}	$\overline{w_{14}}$	w_{15}	$\overline{w_{15}}$
1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	0	1	0	1	0
1	0	0	1	0	1	0	1	1	0
1	0	0	1	0	1	1	0	0	1
1	0	0	1	1	0	0	1	0	1

Table E.4: The table of 6-bit diagnostic sequence pairs ($W_{16} - W_{20}$).

W_{16}		W_{17}		W_{18}		W_{19}		W_{20}	
w_{16}	$\overline{w_{16}}$	w_{17}	$\overline{w_{17}}$	w_{18}	$\overline{w_{18}}$	w_{19}	$\overline{w_{19}}$	w_{20}	$\overline{w_{20}}$
1	0	1	0	1	0	1	0	1	0
0	1	0	1	1	0	1	0	1	0
1	0	1	0	0	1	0	1	0	1
1	0	1	0	0	1	0	1	1	0
0	1	1	0	0	1	1	0	0	1
1	0	0	1	0	1	1	0	0	1

Table E.5: The table of 6-bit diagnostic sequence pairs ($W_{21} - W_{25}$).

W_{21}		W_{22}		W_{23}		W_{24}		W_{25}	
w_{21}	$\overline{w_{21}}$	w_{22}	$\overline{w_{22}}$	w_{23}	$\overline{w_{23}}$	w_{24}	$\overline{w_{24}}$	w_{25}	$\overline{w_{25}}$
1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0
0	1	1	0	1	0	1	0	1	0
1	0	0	1	0	1	0	1	1	0
1	0	0	1	0	1	1	0	0	1
0	1	0	1	1	0	0	1	0	1